# Structured Higher-Order Algorithmic Differentiation in the Forward and Reverse Mode with Application in Optimum Experimental Design

D I S S E R T A T I O N

zur Erlangung des akademischen Grades

Dr. Rer. Nat.
im Fach Mathematik

eingereicht an der
Mathematisch-Naturwissenschaftlichen Fakultät II
Humboldt-Universität zu Berlin

von
**Dipl.-Phys. (ETH) Sebastian F. Walter**

Präsident der Humboldt-Universität zu Berlin:
Prof. Dr. Jan-Hendrik Olbertz

Dekan der Mathematisch-Naturwissenschaftlichen Fakultät II:
Prof. Dr. Elmar Kulke

Gutachter:
1. Prof. Andreas Griewank (PhD)
2. Prof. Dr. Dr. h. c. Hans Georg Bock
3. Prof. Richard D. Neidinger (PhD)

**eingereicht am:** 13. April, 2011
**Tag der mündlichen Prüfung:** 7. Dezember, 2011

**Abstract**

This thesis provides a framework for the evaluation of first and higher-order derivatives and Taylor series expansions through large computer programs that contain numerical linear algebra (NLA) functions. It is a generalization of traditional algorithmic differentiation (AD) techniques in that NLA functions are regarded as black boxes where the inputs and outputs are related by defining equations. Based on the defining equations, structure-exploiting algorithms are derived. More precisely, novel algorithms for the propagation of Taylor polynomials through the QR, Cholesky,- and real-symmetric eigenvalue decomposition are shown. Recurrences for the reverse mode of AD, which require essentially only the returned factors of the decomposition, are also derived. Compared to the traditional approach where all intermediates of an algorithm are stored, this is a reduction from $\mathcal{O}(N^3)$ to $\mathcal{O}(N^2)$ for algorithms with $\mathcal{O}(N^3)$ complexity. $N$ denotes the matrix size. The derived algorithms make it possible to use existing high-performance implementations. A runtime comparison shows that the treatment of NLA functions as atomic can be more than one order of magnitude faster than an automatic differentiation of the underlying algorithm. Furthermore, the computational graph is orders of magnitudes smaller. This reduces the additional memory requirements, as well as the overhead, of operator overloading techniques to a fraction.

It is then demonstrated that the novel algorithms can be used to compute the gradient of the optimum experimental design (OED) objective function in the reverse mode of AD. Since the evaluation of the objective function already requires the evaluation of nested derivatives, this is a nonstandard situation. One can show that the computation of the gradient is only about three times as expensive as the function evaluation itself and hence is in very good agreement with the theoretical bound. The proposed method is a combination of the reverse mode, univariate Taylor polynomial arithmetic and interpolation/polarization techniques. This is a significant improvement over previous work and makes it possibly to apply OED to a larger class of problems. Furthermore, this study discusses how the necessary higher-order derivatives can be computed using Taylor polynomial arithmetic. This is also an improvement over the previously used finite difference schemes. Numerical examples from chemical engineering are used to verify the algorithms.

**Zusammenfassung**

Computerprogramme für wissenschaftliches Rechnen sind oft Algorithmen zur Approximation mathematischer Funktionen. In dieser Arbeit werden Techniken beschrieben, die es erlauben (höhere) Ableitungen und Taylorapproximationen solcher Computerprogramme effizient zu berechnen. Auch inbesondere dann, wenn die Programme Algorithmen der numerischen linearen Algebra (NLA) enthalten. Im Gegensatz zur traditionellen algorithmischen Differentiation (AD), bei der die zugrunde liegenden Algorithmen um zusätzliche Befehlere erweitert werden, sind in dieser Arbeit die Zerlegungen durch definierende Gleichungen charakterisiert. Basierend auf den definierenden Gleichungen werden Strukturausnutzende Algorithmen hergeleitet. Genauer, neuartige Algorithmen für die Propagation von Taylorpolynomen durch die QR, Cholesky und reell-symmetrischen Eigenwertzerlegung werden präsentiert. Desweiteren werden Algorithmen für den Rückwärtsmodus der AD hergeleitet, welche im Wesentlichen nur die Faktoren der Zerlegungen benötigen. Im Vergleich zum traditionellen Ansatz, bei dem alle Zwischenergebnisse gespeichert werden, ist dies eine Reduktion von $\mathcal{O}(N^3)$ zu $\mathcal{O}(N^2)$ für Algorithmen mit $\mathcal{O}(N^3)$ Komplexität. $N$ ist hier die Größe der Matrix. Zusätzlich kann bestehende, hoch-optimierte Software verwendet werden. Ein Laufzeitvergleich zeigt, dass dies im Vergleich zum traditionellen Ansatz zu einer Beschleunigung in der Größenordnung 100 führen kann. Da die NLA Funktionen als Black Box betrachtet werden, ist desweiteren auch der Berechnungsgraph um Größenordnungen kleiner. Dies bedeutet, dass Software, welche Operator Overloading benutzt, weniger Overhead hervorruft und auch weniger Speicher benötigt.

Dann wird demonstriert, dass die neuartigen Algorithmen für die Berechnung des Gradientens der optimalen Versuchsplanung verwendet werden können. Da die Berechnung der Zielfunktion bereits erste Ableitungen erfordert, ist dies keine Standardsituation. Nichtsdestotrotz kann man den Gradienten in etwa dem dreifachen der Zeit der Zielfunktionberechnung erhalten. Dies ist in guter Übereinstimmung mit der AD Theorie. Die angewandte Methode ist eine Kombination des Rückwärtsmodus, univariater Taylorarithmetik und Polarisationsformeln. Dies erlaubt es auch höhere Ableitungen zu berechnen. Somit bieten die Techniken einen wesentlichen Fortschritt gegenüber des bisherig verwendeten Vorwärtsmodus und numerischer Differentiation. Numerische Beispiele aus der chemischen Verfahrenstechnik werden als Testbeispiele für die Verifikation des Gesamtkonzepts verwendet.

## Acknowledgements

I thank my advisors Prof. Andreas Griewank (PhD), Humboldt-Universität zu Berlin, and Dr. Stefan Körkel, Interdisziplinäres Zentrum für Wissenschaftliches Rechnen (IWR), University of Heidelberg, for their excellent support over the last three years. It was a pleasure to work with such important contributers to *algorithmic differentiation* and *model based optimum experimental design*. Their extraordinary expertise provided an optimal starting point and greatly influenced the direction this work took.

I owe an enormous debt of gratitude to Dr. Lutz Lehmann, Humboldt-Universität zu Berlin. He has deeply influenced my perception of mathematics in the many discussions we had and made several contributions to major ideas of the presented work.

The discussions with my collaboration partners Dr. Mario Mommer, Dr. Tilman Barz and Dr. Hergen Schultze within the BMBF project NOVOEXP greatly helped to broaden my horizon and lead to a deeper understanding of the issues one encounters in practice. I also owe gratitude to Jan Albersmeyer, who was not part of the project, but gave nonetheless kind support for his software DAESOL-II. Beyond this collaboration, I'm happy to have met Dr. Bradley Bell, University of Washington. His enthusiastic and result-oriented nature made it pleasant to cooperate with him to create Python bindings to the C++ software CppAD. The cooperation with Dr. Thomas Carraro to create the community website `www.odoe.org` was also a pleasant experience.

For proof reading of some of the parts I thank René Lamour and Nikolai Strogies, both Humboldt-Universität zu Berlin. Last but not least I am grateful for the constant support of my family and my girlfriend Bolor.

This research was partially supported by the Bundesministerium für Bildung und Forschung (BMBF) within the project NOVOEXP (Numerische Optimierungsverfahren für die Parameterschätzung und den Entwurf optimaler Experimente unter Berücksichtigung von Unsicherheiten für die Modellvalidierung verfahrenstechnischer Prozesse der Chemie und Biotechnologie) (03GRPAL3), Humboldt Universität zu Berlin.

# Contents

# 1. Introduction

## 1.1. Problem Description

To explain the scope of this thesis it is necessary to provide an example that shall help the reader to understand the overall problem. A more detailed description follows in the course of the document.

The basic assumption in science and engineering is that physical systems can be approximated by *models* described in mathematical terminology. A computational model can be used to *simulate* the behavior of a physical system under various conditions on a computer. In chemical engineering, many problems can be modeled by differential algebraic equations of the form

$$\begin{aligned} A(t,y,z,p)\dot{y} &= f(t,y,z,u,p) \\ 0 &= g(t,y,z,u,p)\ , \end{aligned}$$

where $y = y(t) \in \mathbb{R}^{N_y}$ and $z = z(t) \in \mathbb{R}^{N_z}$ are state variables, $t \in [0, t_e]$ the time, $u = u(t) \in \mathbb{R}^{N_u}$ some control function and $p \in \mathbb{R}^{N_p}$ parameters. In Chapter 4 one can find a concise discussion of differential algebraic equations. The real numbers are denoted by $\mathbb{R}$. The formulas of $A(t,y,z,p)$, $f(t,y,z,u,p)$ and $g(t,y,z,u,p)$ are found in a *modeling process*, where several assumptions and laws, such as the conservation of mass, are taken into account. Models derived in such a way generally contain *parameters* $p$ which are *given by nature* but for which no numerical values are known. At some point it is therefore necessary to make a connection between model and reality: i.e., to set up an *experiment* and take *measurements* $\eta \in \mathbb{R}^{N_\eta}$. The measurement process is described by a *regression model* which includes a *measurement function*

$$h(y,z,u,p)\ ,$$

also called *measurement model*. It describes how the state is related to the predicted, i.e., simulated, measurements. From the taken measurements one hopes to deduce appropriate values for the parameters $p$.

Basically all measurement processes introduce an additional *statistical error*. Put in other words, one just knows that the true value lies in the vicinity of the observed value. It is, however, possible to define an associated confidence region. I.e., a region believed to contain the true value with a certain probability. The error in the measurement process implies that the deduction of the parameters may also be erroneous. One says that the *error propagates* from the measurements to the parameters. The mathematical details are elaborated in Chapter 5.

It often happens that completely different numerical values for the parameters $p$ can be used in a simulation and, nonetheless, one finds that measurements are explained very well in the given error tolerances. Consider for instance the Diels-Alder reaction which is described by an initial value problem

$$\begin{aligned} \dot{y}_1 &= -k \cdot \frac{y_1 \cdot y_2}{m_{tot}}, & y_1(0) &= q_{y_{a1}} \\ \dot{y}_2 &= -k \cdot \frac{y_1 \cdot y_2}{m_{tot}}, & y_2(0) &= q_{y_{a2}} \\ \dot{y}_3 &= \quad k \cdot \frac{y_1 \cdot y_2}{m_{tot}}, & y_3(0) &= 0\ , \end{aligned}$$

where

$$k = p_{k_1} \exp\left(-\frac{p_{E_1}}{R}\left(\frac{1}{u_T(t)} - \frac{1}{T_{ref}}\right)\right) + p_{k_{kat}} q_{c_{kat}} \exp\left(-p_\lambda t\right) \exp\left(-\frac{p_{E_{kat}}}{R}\left(\frac{1}{u_T(t)} - \frac{1}{T_{ref}}\right)\right) .$$

Additionally, there is a solvent $y_4 = q_{y_{a4}}$ and an algebraic equation $m_{tot} = y_1 M_1 + y_2 M_2 + y_3 M_3 + y_4 M_4$ which describes the conservation of mass. Hence, this model is a differential algebraic equation as described above. In addition to the control function $u_T(t)$, which describes the temperature profile, there are several control variables $q$ such as the catalyst concentration $q_{c_{kat}}$ or the initial values $y_1(0) = q_{y_{a1}}$ and $y_2(0) = q_{y_{a2}}$. The control function $u_T(t)$ is parameterized by finitely many control variables and is here of piecewise linear continuous form. Note that control variables are, for notational simplicity, considered to be globally constant control functions in the ongoing discussion (see Chapter 4.3). The overall model is described in detail in Chapter 7.2. Now, assume that

$$p_{\text{true}} = (p_{k_{\text{cat}}}, p_{E_1}, p_{k_1}, p_{E_{\text{cat}}}, p_\lambda) = (0.01, 60000, 0.1, 40000, 0.25) \tag{1.1}$$

are the true parameters. By true it is meant that if there was no error in the measurement process, the observed measurements would lie exactly on the trajectory

$$h(y, z, p) = \frac{100 \cdot y_3(t) M_3}{y_1(t) M_1 + y_2(t) M_2 + y_3(t) M_3 + q_{ya4} M_4} .$$

However, in practice errors cannot be avoided. In Figure 1.1 one can see the measurement model and simulated measurements $\eta$ at the measurement times $t_{\text{mts}}$. The errorbars indicate the standard deviations.

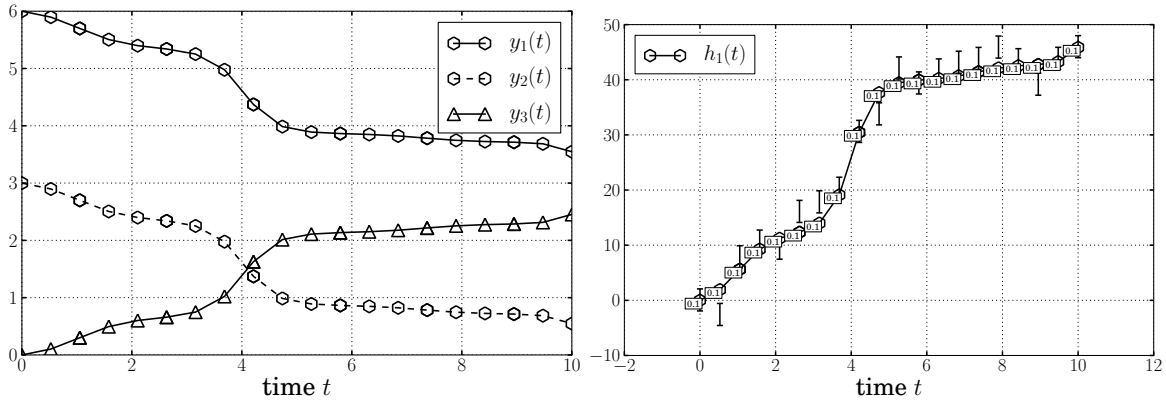



Figure 1.1.: The left plot shows a simulation of the state $y(t; y_0, u, p_{\text{true}})$, where $p_{\text{true}}$ takes the values as defined in (1.1). On the right side one can see the measurement function $h_1(t) := h(y(t, y_0, u, p_{\text{true}}), z, p_{\text{true}})$.

Of course, numerical values for true parameters are not known in advance. The idea is to choose the parameters such that the simulated measurement function $h$ describes the observed measurements well. Unfortunately, this *inverse problem* is not necessarily well-posed. If one uses the numerical values

$$\begin{aligned} p &= (0.01, 6000, 0.1, 40000, 0.25) \\ \text{or} \quad p &= (0.01, 60000, 0.1, 4000, 0.25) \end{aligned} \tag{1.2}$$

as parameters, one obtains an output as shown in Figure 1.2. Comparison to Figure 1.1 shows

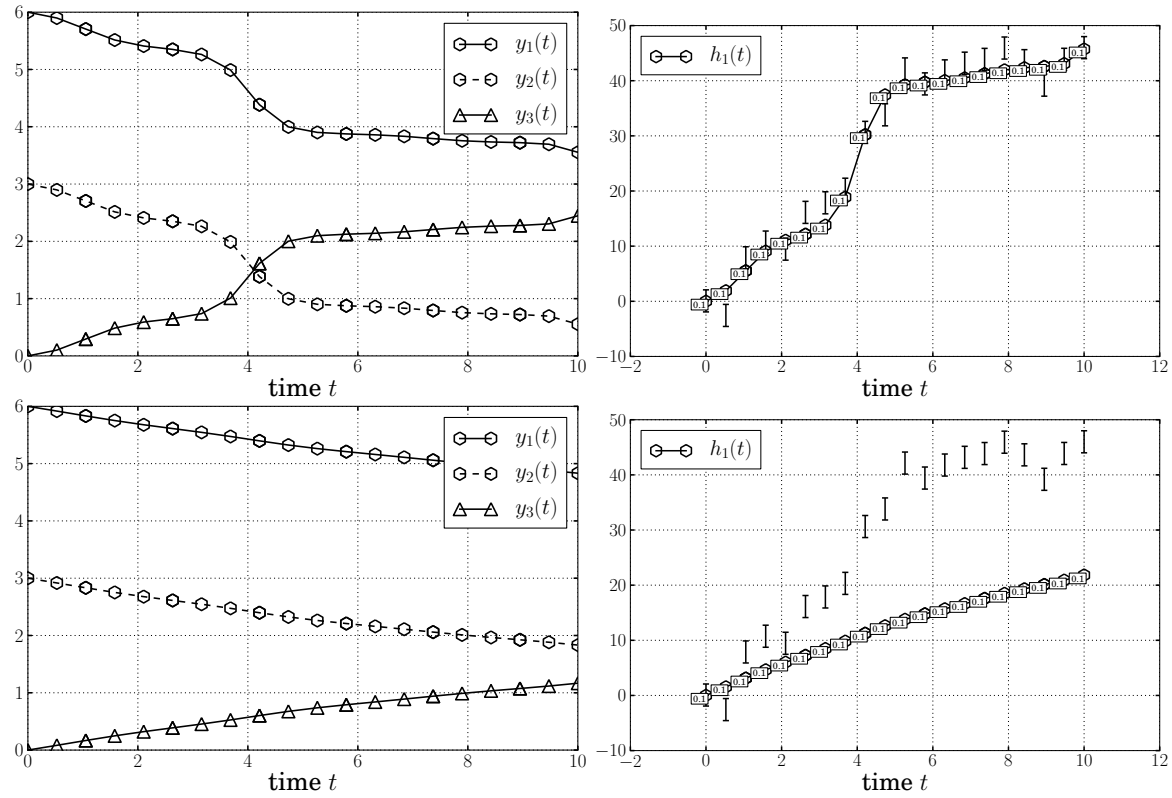



Figure 1.2.: This figure shows on the left the dynamics of a chemical reaction and on the right the measurement function. The error bars indicate the observed measurements and their standard deviations. The two different simulations correspond to the different parameter values of (1.2). As one can see, a change in the second parameter has hardly any effect whereas a change, on the same order of magnitude, to the fourth parameter changes the whole dynamics. Looking closely one can also see that the weight per measurement is 0.1.

that, despite the large change in the fourth parameter, one cannot observe any change in the state and the measurement function. Hence, it would be rather difficult to estimate the fourth parameter since the model is rather insensitive to it. One is tempted to argue that it is simply not important to know the exact value. However, the system may be much more sensitive to the fourth parameter under different conditions. This situation may occur when the scale of a chemical reactor is increased. Or, to give another example, consider a catalytic converter (cat) as used in most cars today. The cat should have a lifetime of many years. Since the development only allows one to run tests in a much shorter time-frame, it may be necessary to estimate for instance the degeneration rate of the catalyst. Already a small deviation of the estimate from the true value may result in a defective cat before the minimal required life expectancy. I.e., although it is difficult to estimate such parameters accurately, it may be important to do so. As these two examples show, it can be of central importance to find numerical estimates $\hat{p}$ of the parameters $p$ with *small confidence region*.

For scalar parameters $p \in \mathbb{R}$, the confidence regions are generally confidence intervals described by the standard deviation. E.g., $p = 0.01 \pm 0.001$ means that the standard deviation is 0.001. Therefore, the probability that the true parameter lies in the interval $[0.01 - 0.001, 0.01 + 0.001]$ is about 68.2%, granted the error is normally distributed. More generally, the parameters can be correlated and the confidence region is approximated by an ellipsoid, which can be described by a covariance matrix. A discussion of confidence regions and estimates can be found in Chapter 5.1.

It is well known that one can improve the confidence region by taking the average of several experiments (performed under the same conditions). Then the error scales with $\frac{1}{\sqrt{N_{\text{rep}}}}$ under certain conditions on the probability distribution, where $N_{\text{rep}}$ is the number of repetitions of the experiment. To gain one digit of accuracy, it is necessary to perform 100 times as many runs. Since in chemical engineering an experiment can be very costly –in time and money–, this scaling is a major limiting factor. Fortunately, there is an additional "degree of freedom" that influences the errors: the experimental setup. In this document, it is assumed that the experimenter can adjust control functions $u(t)$ (this includes also control variables) and can decide when to measure. The process of the experimental design optimization, and its related mathematical theory, is called *optimum experimental design* (OED) or *optimal design of experiments* (ODOE).

An optimization can drastically reduce the size of the confidence region. Consider again the Diels-Alder reaction from above. A random choice of the experimental setup results in the covariance matrix

$$C = \begin{pmatrix} 5.258e-02 & -1.126e-02 & -3.244e-01 & 8.302e-02 & 1.461e+00 \\ -1.126e-02 & 4.810e-03 & 1.478e-01 & -2.096e-01 & -1.818e-01 \\ -3.244e-01 & 1.478e-01 & 5.298e+00 & -6.964e+00 & -3.760e+00 \\ 8.302e-02 & -2.096e-01 & -6.964e+00 & 1.701e+01 & -7.199e+00 \\ 1.461e+00 & -1.818e-01 & -3.760e+00 & -7.199e+00 & 5.036e+01 \end{pmatrix}.$$

After the optimization one obtains

$$C = \begin{pmatrix} 6.680e-04 & -1.784e-04 & -7.663e-05 & 3.753e-05 & 1.345e-04 \\ -1.784e-04 & 9.165e-05 & 3.293e-05 & -3.269e-05 & -3.239e-06 \\ -7.663e-05 & 3.293e-05 & 1.042e-03 & -7.164e-04 & -6.737e-05 \\ 3.753e-05 & -3.269e-05 & -7.164e-04 & 8.188e-04 & 2.201e-03 \\ 1.345e-04 & -3.239e-06 & -6.737e-05 & 2.201e-03 & 1.911e-02 \end{pmatrix}.$$

One can see that the entries are much smaller after the optimization. A more detailed interpretation of the confidence matrix can be found Chapter 5 and Chapter 6.2. Note that the element $C_{22}$ is much smaller than $C_{44}$ in the initial experimental setup: a fact that was deduced heuristically in the above discussion. In both cases, the covariance matrix has been scaled according to $C := VCV$ in a post-processing step, where $V = \text{diag}(1/p)$ has the inverses of the parameter values on its main diagonal. This rescaling makes it easier to interpret the shown covariance matrices. Consider, for instance, the first parameter of the unoptimized experiment. It has a variance of $5.258e-02$ and therefore the standard deviation is its square root 0.230. Neglecting the correlations and approximation errors, one can state that the true parameter $p_1$ is believed to lie in $1 \pm 0.230$ with a probability of 68.2%. In Figure 1.3 one can see what the initial and the optimized experiment look like.

When there are only a few control variables, one can try to find an appropriate experimental design by simulating several scenarios. However, as the number of controls is increased, such trial and error quickly becomes infeasible since only a comparatively small part of the search space can be investigated. It is therefore imperative to use a computer and perform a numerical optimization. Because just the simulation of a model, in particular when it stems from a partial differential equation, can be time consuming, it is necessary to use good optimization algorithms whose iterates converge super-linearly to a local minimum. For nonlinear programs of sufficiently smooth functions, there exist reliable algorithms and implementations in software. They require at least the gradient of the objective function and potentially also Hessian-vector products.

In Chapter 6 it is shown that confidence region can be derived, to first order, by linear error propagation through the linearized least squares solution operator. In the process, it is necessary

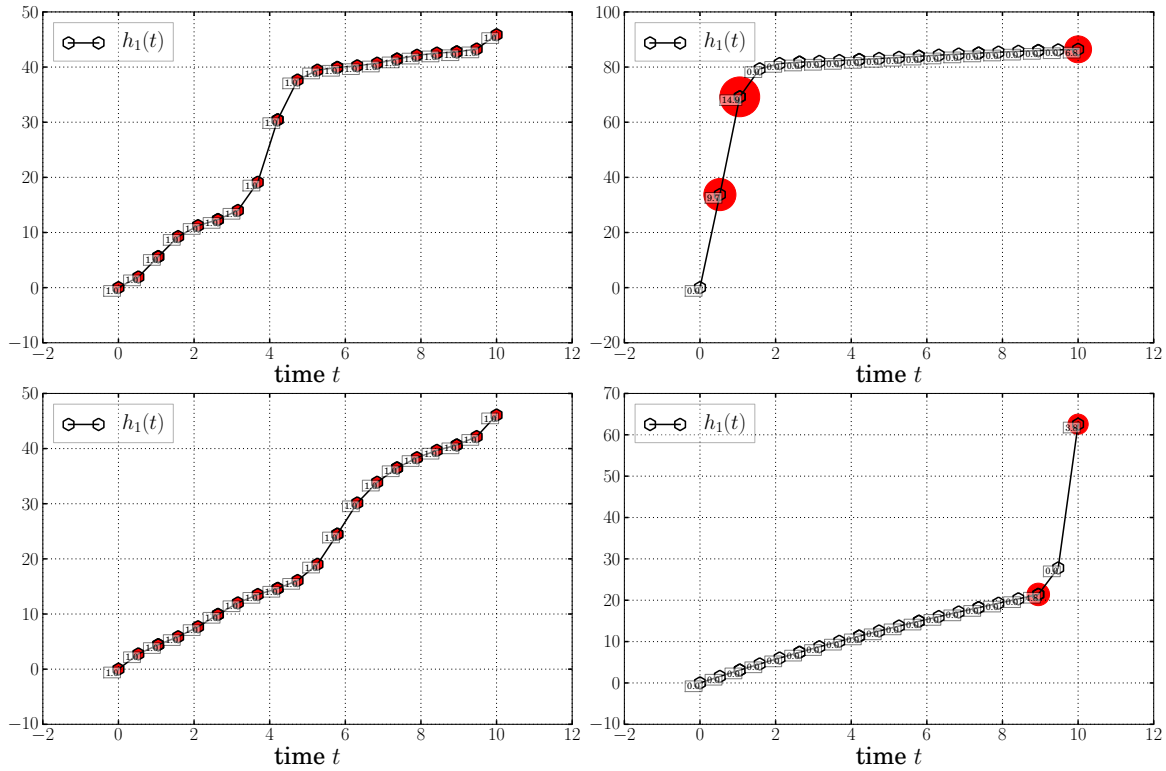



Figure 1.3.: On the left, the measurement function of the initial experimental design. For every measurement time the weight is 1. On the right one can see the optimized experimental design. Apparently it is advantageous to concentrate the measurements at some special measurement times. The total number of measurements is 40 and the weight of a measurement is illustrated by a (red) dot.

to assemble the matrices

$$J_1 := \Sigma^{-1} W(w) \frac{\partial h}{\partial p} \quad \text{and} \quad J_2 := \frac{\partial F_2}{\partial p}$$

from evaluations of the measurement function $h(y, z, u, p)$ at certain measurement times. $F_2$ is some constraint function as explained in Chapter 5. When these matrices have been evaluated, they are used to compute the covariance matrix

$$C = K Q_2^T \left( Q_2 J_1^T J_1 Q_2^T \right)^{-1} Q_2 K^T \,,$$

where $Q_2$ spans the nullspace of $J_2$ and can be computed using the QR decomposition of $J_2$, i.e.,

$$J_2^T = (Q_1^T, Q_2^T) \begin{pmatrix} L^T \\ 0 \end{pmatrix} .$$

A derivation can be found in Chapter 5.2.3. The covariance matrix itself is input to another function, for instance,

$$\Phi(C) := \operatorname{tr}(C) \,,$$

$\det(C)$ or $\operatorname{eigh}(C)$ (symmetric eigenvalue decomposition). In other words, apart from the simulation of the underlying dynamics, algorithms from numerical linear algebra are a central part of the objective function evaluation. Robust OED formulations may even require derivatives of

third and higher order. For instance, Bock et al. [2004] introduce the objective function

$$\Phi_{\text{robust}} := \Phi + \gamma \sqrt{\frac{\partial \Phi}{\partial p}^T \Sigma^{-2} \frac{\partial \Phi}{\partial p}} \,,$$

where $\Sigma^2$ is the covariance matrix of the parameters $p$ and $\gamma \in \mathbb{R}$ a confidence quantile.

## 1.2. Related Work and Scope

As discussed in the previous section, a key requirement for the efficient optimization of experimental designs is the **efficient and numerically accurate evaluation of higher-order derivatives of potentially very large computer programs containing numerical linear algebra functions**. Existing techniques from algorithmic differentiation can, in principle, be used to evaluate the desired derivatives. The current state of the art is described in the book by Griewank and Walther [2008]. However, when such techniques are implemented in software, the resulting code may require too much memory, may introduce too much overhead or may even give wrong results. See Chapter 3 for the discussion. Obviously, one would like to possess a framework that meets the requirements (stated above in bold font). This is the central theme of this thesis.

Chapter 2 gives an overview of existing work and introduces the notation: In Chapter 2.1 it is explained how large computer programs can be considered as a computational graph or as a composite function (c.f. Griewank [2003], Gay [1991]). Chapter 2.2 summarizes well-known results from calculus. Based on this well-known theory, the algebraic class of univariate Taylor polynomials (UTP) (Chapter 2.3) is introduced. This is motivated by the work of Berz [1996] and Shamseddine [1999] who investigated the Levi-Civita field. *Interpolation methods* are employed to evaluate (mixed) partial derivatives of the form

$$\frac{\partial^{|\mathbf{i}|}}{\partial z_1^{i_1} \partial z_2^{i_2} \dots \partial z_K^{i_K}} f(x + Sz) \bigg|_{z=0} \,,$$

where $f : \mathbb{R}^N \to \mathbb{R}$ and $z = (z_1, \dots, z_K)^T \in \mathbb{R}^K$. Important contributions are the papers by Neidinger [2005] and Griewank et al. [2000, 2009]. Several interpolation techniques are explained and illustrated by examples. For instance, it is demonstrated how it is possible to compute *all partial derivatives up to a certain degree* or *just a few off-diagonal elements of the Hessian*. The traditional reverse mode of algorithmic differentiation is explained in Chapter 2.5. It is shown how it can be generalized to the reverse mode using Taylor polynomials by *pullbacks of linear forms*. The section explains how this concept is related to existing approaches as described by Christianson [1991] or Griewank and Walther [2008].

After the discussion of these more or less well-known results, they are generalized to a treatment of numerical linear algebra functions in Chapter 3. The discussion is mostly motivated by the work of Giles [2008, 2007], who was the first to collect several results from the literature and describe them in the standard AD notation. There are many treatises of matrix calculus, e.g., the textbooks by Magnus and Neudecker [1999], Seber [2007], Healy [2000], Schott [1997]. These references do not put their results in an algorithmic context and rather provide symbolic identities. Much less literature exists on Taylor polynomial arithmetic applied to NLA functions, e.g., by Vetter [1973]. But so far, apparently only Phipps [2003] has described univariate Taylor polynomial arithmetic applied to the NLA functions `dot, solve, inv` in the combined forward/reverse mode for use in a Taylor series integrator for differential algebraic equations. Even less work has been done on the differentiation of matrix factorizations. Smith [1995] differentiated the Cholesky algorithm by hand. For the eigenvalue decomposition, there exist results for univariate Taylor polynomial arithmetic by Andrew and Tan [1998] as well as van der Aa

et al. [2007], but they neither mention the reverse mode of AD nor do they put their algorithms into a global computational context.

The first central goal of this thesis is to combine these concepts and ideas in a **unified framework**. A second goal is the derivation of **novel algorithms** for the

- univariate Taylor polynomial arithmetic
- and for the reverse mode

of **numerical linear algebra functions** such as the Cholesky ( Chapter 3.6), QR (Chapter 3.7 resp. Chapter 3.8) and the real symmetric eigenvalue decomposition (Chapter 3.9). It is shown that the memory requirement (in the reverse mode) of the proposed algorithms is essentially the amount of memory required to store the results. A basic runtime comparison is performed in Chapter 3.11. One observes that the treatment of NLA functions as atomic can be up to two orders of magnitudes faster than an automated differentiation of the algorithm.

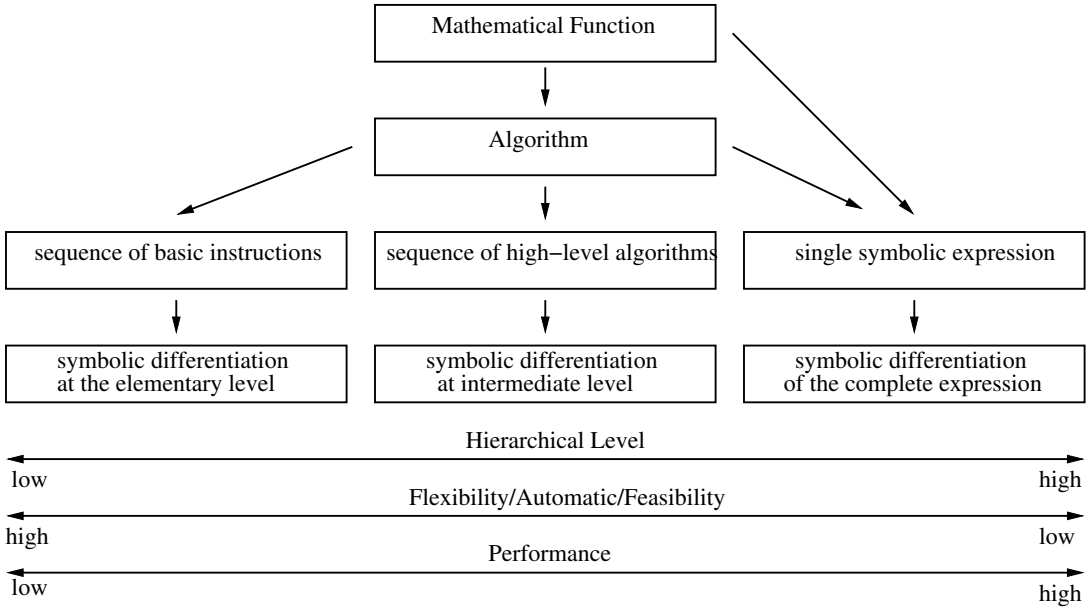



Figure 1.4.: This figure illustrates that algorithmic differentiation is, on some level, of a symbolic nature. In a fully symbolic treatment, the overall algorithm would be considered as one symbolic expression whereas the traditional AD approach regards the algorithm as a sequence of elementary functions. In this thesis, an intermediate possibility is advocated. I.e., the overall computation is regarded as a sequence of high-level functions for which structure exploiting algorithms are derived.

The discussion of the AD techniques is rather general and is therefore applicable to many problems in science and engineering. In contrast, the following discussion seeks to apply the new algorithms in *model based optimum experimental design* (OED). As highlighted in the previous section, the optimization of experiments can result in large improvements in the accuracy of parameter estimates. There is much literature on statistical aspects, in particular of linear models, see, for instance, the work by Mead [1990] or Pukelsheim [1993]. There is a growing interest in a treatment of nonlinear models. This trend is reflected in workshops, minisymposia on conferences and publications, e.g., Winterfors and Curtis [2008]. A good deal of the publications treat questions as they arise in drug testing: given a certain number of test subjects, insert a certain amount of a drug and test whether it has a (positive) effect. These models from pharmacokinetics are typically relatively simple algebraic or ordinary differential equa-

tions. The focus is often on the statistical aspects and not so much the efficient optimization. See, for example, the thesis by Waterhouse [2005] or the software [Duffull et al., 2009–2010].

For model based OED with application in chemical engineering, there exists also a variety of publications. The survey paper by Franceschini and Macchietto [2008] strives to provide an overview of the current state of the art in model based OED from a chemical engineer's point of view. Rasch and Bücker [2010] provide an overview of available software. Directly related to this thesis is the work of Körkel [2002]. Using results from Bock et al. [2007b], the discussion is simplified and combined with the theory of AD. It is demonstrated that one can use the combination of univariate Taylor polynomial arithmetic, interpolation/polarization and the reverse mode to compute all desired derivatives. I.e., also derivatives of higher order. More importantly, it is demonstrated that it is possible to **compute the gradient of the objective function in a time which is a small multiple of the time required to compute the objective function itself**. This is in agreement with the theory of AD. Hence, it is possible **to optimize experimental designs when there are many control variables** $q$. Since the proposed framework in the first part of the thesis is of a general nature, it will be possible to define other, more elaborate, objective functions. Examples of industrial relevance are used to check the results. Figure 1.5 shows a graphical representation of the general structure of the document.

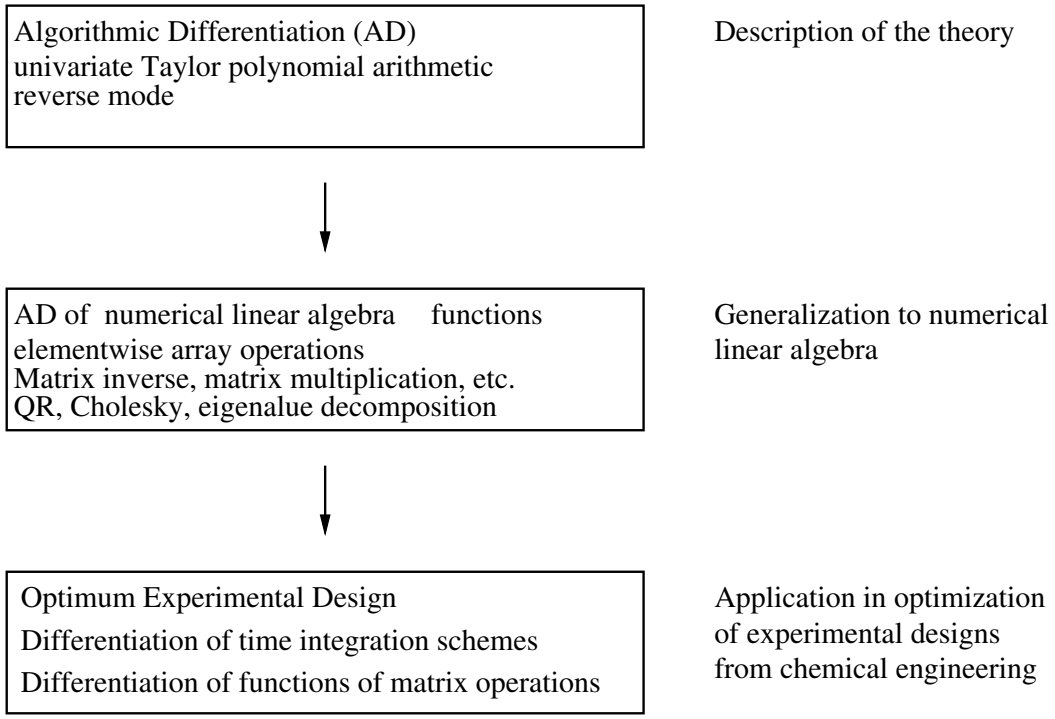



Figure 1.5.: General outline of the thesis.

## 1.3. Publications and Software

Several publications related to the contents of this thesis have been submitted.

- *Algorithmic differentiation in Python using AlgoPy*, special issue on *Scientific Software* in the Journal of Computational Science. Currently in revision.
- *On evaluating higher-order derivatives of the QR decomposition of tall matrices with full*

*column rank in forward and reverse mode algorithmic differentiation* , S. F. Walter, L. Lehman., R. Lamour, in revision, Optimization Methods and Software

- *On the Efficient Evaluation of Higher-Order Derivatives of Real-Valued Functions Composed of Matrix Operations*, S. F. Walter, proceedings of the High-Performance Scientific Computing conference, Hanoi 2009

Also notable is the creation of the community website `www.odoe.org`. It provides a central knowledge hub for practitioners and scientists.

At some point it is necessary to depart from theoretical considerations and actually perform the simulation or the optimization on a computer. Then it is necessary to translate mathematical concepts into a language a digital computer can understand. This transition is often a long, cumbersome and error-prone process. In the course of this thesis, several software tools have been implemented in the programming languages C/C++ and Python. They can be grouped into three classes:

1. Bindings to existing C/C++ and Fortran code:
   - The Python bindings **PyAdolc** to ADOL-C (C++) [Griewank et al., 1999], available at www.github.com/b45ch1/pyadolc.
   - The Python bindings **pyccpad** to CppAD (C++) [Bell, 2010], available at www.github.com/b45ch1/pycppad.
   - The Python bindings **PySolvIND** to SolvIND (C++) [Albersmeyer and Kirches, 2007–]. The software includes also an interface to PyAdolc and allows the differentiation of model functions written in Fortran.
2. New software useful for the evaluation of derivatives in C and Python:
   - The Python tool **AlgoPy** supports the forward and reverse mode of algorithmic differentiation of Python code containing numerical linear algebra functions and vectorized operations. It is available at www.github.com/b45ch1/algopy. The focus is not the efficient evaluation but its versatility and ease of use.
   - The ANSI-C tool **taylorpoly**. It is the attempt to create simple building blocks similar to LAPACK to make a rapid development of efficient codes by hand possible, but could also be used as part of an automated process. It is based on the the insights gained when AlgoPy was developed and strives to provide a library of algorithms which combine high-performance and versatility.
3. All the pieces are then put together in the prototype **EasyOdoe**. It is designed to optimize experimental designs and has been used for all numerical experiments in this thesis. Much of the design is strongly influenced by the software tool VPLAN written by Körkel [2002]. EasyOdoe uses PyAdolc and Tapenade (c.f. Hascoët and Pascual [2004]) to evaluate the derivatives of the model functions $A, f, g$. The integration in forward/reverse univariate Taylor polynomial arithmetic is done by PySolvIND. A standard nonlinear optimizer is used to optimize the design.

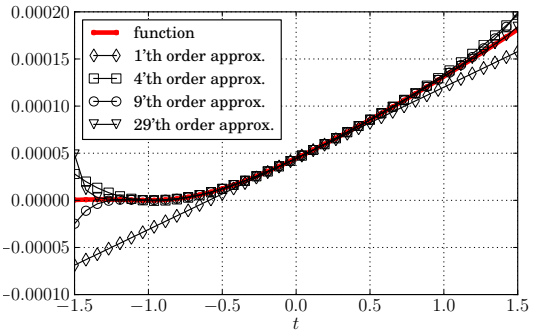
Despite the fact that the author spent most of his time in the last three years to write code, the focus of this thesis is **not** a manual thereof. Consult the software documentation for a more detailed description. Nonetheless, the software did the number crunching and may be of interest for those who are confronted with similar problems and want to profit from the available implementations. To give an idea how the new algorithms can be applied, consider the following two examples in Figure 1.6 and Figure 1.7.

The first shows a Python program containing a loop where at each step one row of a temporary matrix is computed. The resulting temporary matrix is consecutively used as an input for

numerical linear algebra routines. As one can see in Figure 1.6 it is possible to compute Taylor series expansions to high order through the overall program. In Figure 1.8 the computational graph of the function evaluation is depicted. One can see that the linear algebra functions are treated as atomic. The computational graph is thus of very moderate size.

The second example shows a Python program of an objective function $f : \mathbb{R}^{M\times M} \to \mathbb{R}$. In Figure 1.7 one can see the result of the optimization with additional box constraints in form of a cylinder. It demonstrates that vectorized operations allow a compact representation of the problem and also shall serve as an example where the reverse mode of AD is necessary due to the high dimensionality $M^2$ of the domain.

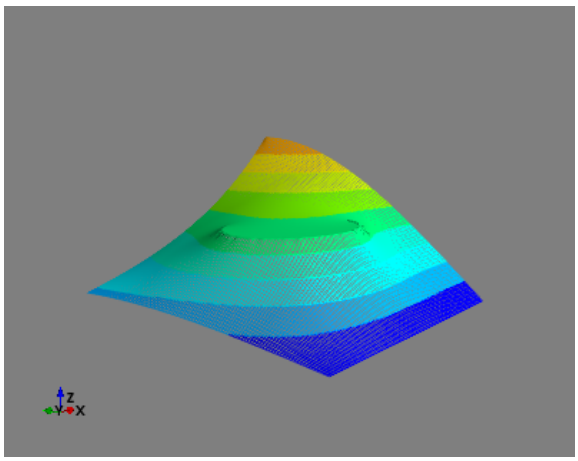
As one can see, Python is very pseudo-code-like. It should be possible for the interested reader to understand the code examples without consulting the manuals. Therefore, several code snippets are shown to illustrate certain theoretical results.




```
1  import algopy; import numpy
   from algopy import zeros, dot, qr, eigh
   Nts = 10
   def eval_f(x):
       tmp = zeros((Nts,x.size),dtype=x)
6      tmp[0] = x
       for nts in range(1,Nts):
           x[0] -= 0.001*x[0]*x[1]
           x[1] += 0.001*x[0]*x[1]
           tmp[nts] = x
11
       Q,R = qr(tmp)
       l = eigh(algopy.dot(R.T,R))[0]
       return l[0]/l[1]

16 def eval_x(t):
       return t*numpy.array([1,2])
              +numpy.array([3,2])
```

Figure 1.6.: The Python code on the right shows a prototypical example that illustrates the type of program appearing in optimum experimental design. I.e., at first a loop akin to an integration scheme, the storage of the state trajectory in a matrix and finally numerical linear algebra functions applied to the matrix. All operations can be performed in univariate Taylor polynomial arithmetic. On the left, a Taylor series expansion is shown. One can see that it is a local approximation of the function.



```
import algopy; import numpy;
def O_tilde(u):
    return 1 + \
    algopy.sum((u[1:,1:]-u[0:-1,0:-1])**2+
          (u[1:,0:-1] - u[0:-1, 1:])**2)/4.
```

Figure 1.7.: Minimal surface after optimization with additional box constraints and boundary values. It illustrates that it is possible to compute the gradient of an objective function in a time which is a small multiple of the time required to compute the function itself. This is necessary in this case since there are $M^2 = 100^2 = 10000$ independent variables.

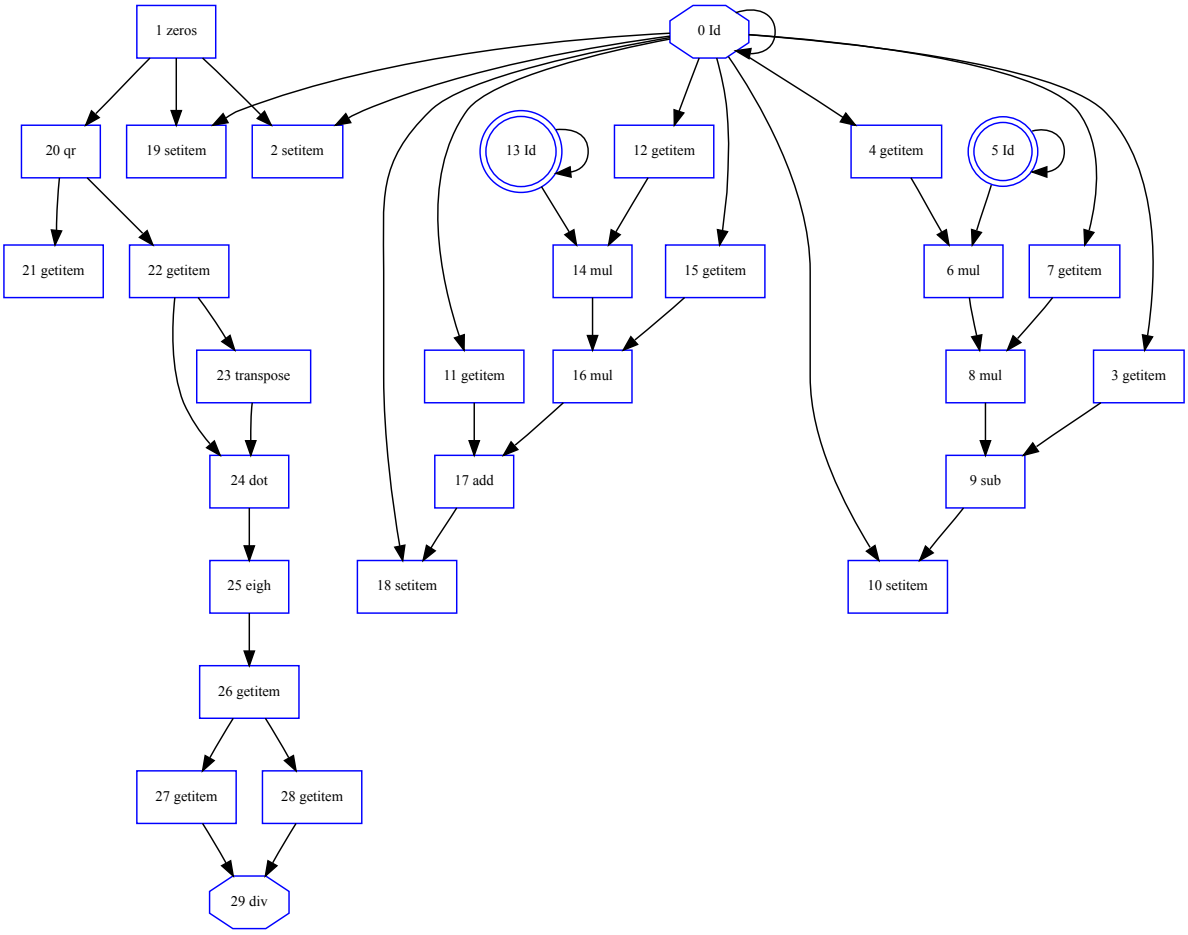



Figure 1.8.: Computational graph of Listing 1.6 but with Nts = 2 to fit the graph on one page. One can see that there are many operations on a mutable data structure (getitem, setitem) as well as several numerical linear algebra functions. Observe that since the numerical linear algebra functions are regarded as atomic functions the computational graph is of only moderate size.

# 2. Algorithmic Differentiation

There is no strict definition of *Algorithmic Differentiation* (AD). Rather, the general consensus is that AD is neither finite differences nor symbolic differentiation. On the AD community website it is defined as follows:

> *"Automatic Differentiation (AD) is a technology for automatically augmenting computer programs, including arbitrarily complex simulations, with statements for the computation of derivatives, also known as sensitivities."*, from Autodiff.org

Hence, a variety of related methods are referred to as *AD techniques*: for instance, univariate/multivariate Taylor arithmetic, the forward mode, the reverse mode, hierarchical methods (c.f. Bischof and Haghighat [1996], Bücker [2002]), vertex elimination and edge elimination (c.f. Naumann [2004, 2009]), just to name the most important. Checkpointing techniques are also added to the list since they reduce the memory requirement of the reverse mode. As Griewank [1992] has shown, they can reduce the memory footprint from $\mathcal{O}(L)$ to $\mathcal{O}(\log(L))$, where $L$ is the number of instructions. Since AD is to a large extent, at least on the level of elementary functions, of symbolic nature, and because the notion of symbolic differentiation is also not explicitly defined, there is sometimes confusion how a (new) technique should be called. For instance, Guenter [2007] calls a technique "efficient symbolic differentiation" which is basically the reverse mode accumulation of symbolic quantities with a subsequent elimination of common subexpressions akin to edge elimination.

Since AD is specifically not symbolic differentiation (SD), it is necessary to describe what is meant by *symbolic computation* and *symbolic differentiation* in particular. In symbolic computations all quantities are symbols that form expressions. Expressions can be transformed by certain substitutions. For instance, $\sin^2(x) + \cos^2(x)$ can be substituted by the symbol 1. A symbolic computation is a sequence of substitutions. Consider the function

$$\begin{aligned} f : \mathbb{R}^2 &\to \mathbb{R} \\ (x_1, x_2) &\mapsto y = \sin(x_1 + \cos(x_2)x_1) \ . \end{aligned}$$

The symbol $y$ is a placeholder for $\sin(x_1 + \cos(x_2)x_1)$, where $\sin, \cos, x_1, x_2$ are symbols themselves. *Symbolic differentiation* is a special type of substitution where $\frac{\mathrm{d}f(g(x))}{\mathrm{d}x}$ can be substituted by $\left.\frac{\partial f(y)}{\partial y}\right|_{y=g(x)} \frac{\mathrm{d}g(x)}{\mathrm{d}x}$. After several of such symbolic substitutions one arrives at the symbolic gradient

$$\begin{pmatrix} \frac{\mathrm{d}f}{\mathrm{d}x_1} \\ \frac{\mathrm{d}f}{\mathrm{d}x_2} \end{pmatrix} = \begin{pmatrix} \cos(x_1 + \cos(x_1)x_1)(1 + \cos(x_2)) \\ -\cos(x_1 + \cos(x_1)x_1)\sin(x_2)x_1 \end{pmatrix} \ .$$

One can see that the two elements of the symbolic gradient share subexpressions. The process of using the expression $\sin(x_1 + \cos(x_2)x_1)$ and its transformation to the symbolic gradient is what is meant by symbolic differentiation in this document.

In contrast to SD, algorithmic differentiation works with the algorithmic implementation of a function. The algorithms can be described in several forms, e.g., in the three-part form or state space representation (Section 2.1). Some authors use *automatic differentiation* or *computational differentiation* as synonyms for algorithmic differentiation. Arguably, "automatic" is a bit misleading since not all aspects of the procedure can be automatized. Or if they could, it

may be computationally expensive since the computer would need a thorough understanding of a program's purpose. For instance, in the computation of directional derivatives of eigenvalues the function evaluation becomes non-unique when there are repeated eigenvalues. At this point, the algorithm does not include all necessary information to describe the function and additional knowledge has to be taken into account. Other issues arise when an algorithm follows different program branches for different regions of the domain. Algorithmic implementations often contain tricks and hacks that return the correct function evaluation but are unsuitable for an automatic differentiation. An example are lookup tables and certain program branches. See Chapter 3 for examples as they appear in numerical linear algebra functions.

The following sections are a summary of the most important techniques in AD which are of relevance for the later discussion. Of fundamental importance is the *description of a mathematical function as a sequence of algorithms*. In Chapter 2.1 it is explained how the algorithms are chained together, defining the *model of computation* used throughout this thesis. After that it is shown in Chapter 2.3 how univariate Taylor series expansions of mathematical functions can be computed easily by decomposing the overall computation in many small subtasks. The basic idea is that one can define an algebraic class of univariate Taylor polynomials. This shares many ideas with the generalization of real arithmetic to complex arithmetic. For this reason it is also briefly discussed how the algebraic class can be endowed with an order and topological structure. The discussion is motivated from the work by Berz [1996] and Shamseddine [1999]. In Chapter 2.4 it is shown how polarization identities can be used in combination with the univariate Taylor polynomial arithmetic to compute partial derivatives and derivative tensors. The traditional reverse mode of AD is briefly described in Chapter 2.5. In contrast to existing literature, the reverse mode is introduced by definition of a *linear form* and the concept of the *pullback*. This generalization makes it possible to describe what is meant by the *combination of univariate Taylor polynomial arithmetic and reverse mode*. The differences between symbolic, numerical and algorithmic differentiation are highlighted in Chapter 2.7.

## 2.1. Model of Computation

One has to make a distinction between mathematical functions $F$ of the form

$$\begin{aligned} F : \mathbb{R}^N &\to \mathbb{R}^M \\ x &\mapsto y = F(x) \ . \end{aligned}$$

and algorithms that can evaluate $F$. From the mathematical point of view, a function is a relation that associates with each $x \in \mathbb{R}^N$ exactly one $y \in \mathbb{R}^M$. In most cases of relevance in scientific computing, a function $F$ is either algorithmically defined or as solution of an implicit system. Since it is usually possible to solve such implicit systems algorithmically, one comes to the conclusion that the function is defined by an algorithm. Contrary to the function, which is uniquely defined by its mapping $x \mapsto y$, there are many algorithms for some given function. For instance, the exponential function $y = \exp(x)$ can be defined as the solution to the differential equation $y'(x) = y(x)$ and two possible recurrences are

$$\exp(x) = \lim_{D \to \infty} \sum_{d=0}^{D-1} \frac{1}{d!} x^d$$

or

$$\exp(x) = \lim_{d \to \infty} \left(1 + \frac{x}{d}\right)^d \ .$$

From this example one can make two observations:

1. there is no one-to-one correspondence between function and algorithm

2. functions such as $\exp(x)$ can be evaluated as a sequence of additions and multiplications.

Since in the end one is interested in an algorithm that evaluates derivatives, it is therefore the question whether to *augment the algorithm*, i.e., a transformation of the program, or if the structure of the function should be exploited to derive a new algorithm suitable for the evaluation of derivatives. However, no matter which method is applied, *the overall computer program is a sequence of algorithms* which evaluate some function. It is therefore mandatory to introduce a description of the overall program.

### 2.1.1. Three-Part Form

Given $x \in \mathbb{R}^N$, an algorithm for $F(x)$ has to terminate after finitely many steps. Hence, for every $x \in \mathbb{R}^N$ the algorithm is described by its sequence of operations. Often, this sequence is invariant under changed inputs. It is therefore possible to describe the algorithm in the so-called *three-part form* [Griewank and Walther, 2008, Griewank, 2003]

$$\begin{aligned} v_{n-N} &= x_n & n &= 1, \dots, N \\ v_l &= \phi_l(v_{j \prec l}) & l &= 1, \dots, L \\ y_{M-m} &= v_{L-m} & m &= M-1, \dots, 0\ , \end{aligned} \tag{2.1}$$

where $L$ is the number of calls of elementary functions $\phi_l$ during the computation of $F$. Running indices $(n, m, l)$ use the same letter as the boundary values $(N, M, L)$ to make the notation easier to read. The $x_n$ are called the *independent variables* and $y_{M-m}$ the *dependent variables*. The sequence of $\phi_l$ is called the *trace* of the function evaluation. The result of $\phi_l$ is denoted $v_l$ and $v_{j \prec l}$ is the tuple of arguments of $\phi_l$. Other authors, e.g., Bartholomew-Biggs et al. [2000], call this representation a *Wengert list*:

**Definition 2.1.1** (pushforward)**.** The action of advancing one step in the computation, i.e., the computation of (the numerical value)

$$v_l = \phi_l(v_{j \prec l})$$

is called *pushforward*. $v_{j \prec l}$ denotes the tuple of all arguments of $\phi_l$. Similarly, evaluating the symbolic expression

$$x \in \mathbb{R}^N \mapsto y = F(x)$$

to a numerical value $y \in \mathbb{R}^M$ is called the *pushforward of the function* $F$.

**Example 2.1.1.** To give a simple example consider the function $f : \mathbb{R}^2 \to \mathbb{R}$, $x \mapsto y = f(x) = \sin(x_1 + \cos(x_2) \cdot x_1)$. Note that functions mapping to the real numbers are denoted $f$ instead of $F$. Its computational trace resp. three-part form is

| | | | | | |
|---|---|---|---|---|---|
| independent | $v_{-1}$ | $=$ | $x_1$ | $=$ | $3$ |
| independent | $v_0$ | $=$ | $x_2$ | $=$ | $7$ |
| | $v_1$ | $=$ | $\phi_1(v_0)$ | $=$ | $\cos(v_0)$ |
| | $v_2$ | $=$ | $\phi_2(v_1, v_{-1})$ | $=$ | $v_1 v_{-1}$ |
| | $v_3$ | $=$ | $\phi_3(v_{-1}, v_2)$ | $=$ | $v_{-1} + v_2$ |
| | $v_4$ | $=$ | $\phi_4(v_3)$ | $=$ | $\sin(v_3)$ |
| dependent | $y$ | $=$ | $v_4$ | | |

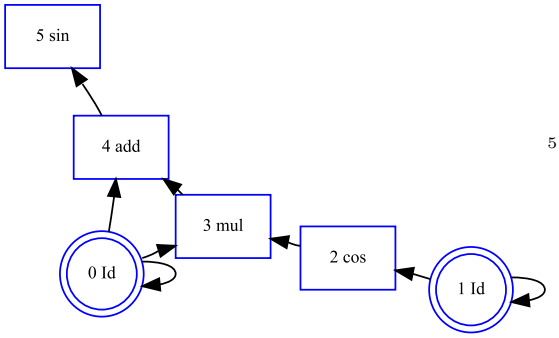
i.e., $L = 4$. Note that for instance $v_{j \prec 2} = (v_1, v_{-1})$. After each step a numerical value $v_l$ is computed. In symbolic computation not the numerical value would be stored in $v_l$ but an expression. This fundamental difference between symbolic and numerical computations explains also the difference between symbolic differentiation and algorithmic differentiation as discussed in Chapter 2.7.1.

### 2.1.2. Representation as Computational Graph

Each instruction $\phi_l$ in the three-part form can be represented by a node in a directed acyclic graph. There is an edge from $\phi_i$ to $\phi_j$ if and only if $j \prec i$. That means that at least one element of the tuple $v_j = \phi_j(v_{k \prec j})$ is an argument of the function $\phi_i$. It is convenient to make no distinction between variables and functions. E.g., a function is formulated as

$$(G \circ F)(x) \Leftrightarrow (G \circ F \circ x)() \ .$$

In that regard, $x : \emptyset \to X$ is a function mapping from the empty set $\emptyset$ to $x() \in X$. This allows one to represent the complete description of a function as a graph containing only function nodes. This can be seen in Figure 2.1.




```
from algopy import *
cg = CGraph()
x1 = Function(3.); x2 = Function(7.)
y = sin(x1 + cos(x2)*x1)
cg.plot('cgraph.svg', method='neato')
```

Figure 2.1.: The computational graph of $y = \sin(x_1 + \cos(x_2) \cdot x_1)$. The integer shown in each node is the step in the computational sequence. E.g., the node "3 mul" means that the third instruction in the computation is a multiplication.

It is also possible to have functions with side effects in the computational graph. However, as one can see in Figure 2.2, the operations on mutable data structures lead to a computational graph of relatively complex nature. In particular, the graph structure does not represent which operations can be performed in parallel; a feature that will likely be of importance on CPUs with many cores. One should note that evaluating expressions based on a computational graph introduces additional pointer arithmetic which results in overhead. I.e., an AD tool should either try to make the graph as small as possible or translate the graph into another representation, e.g., into a sequential tape.

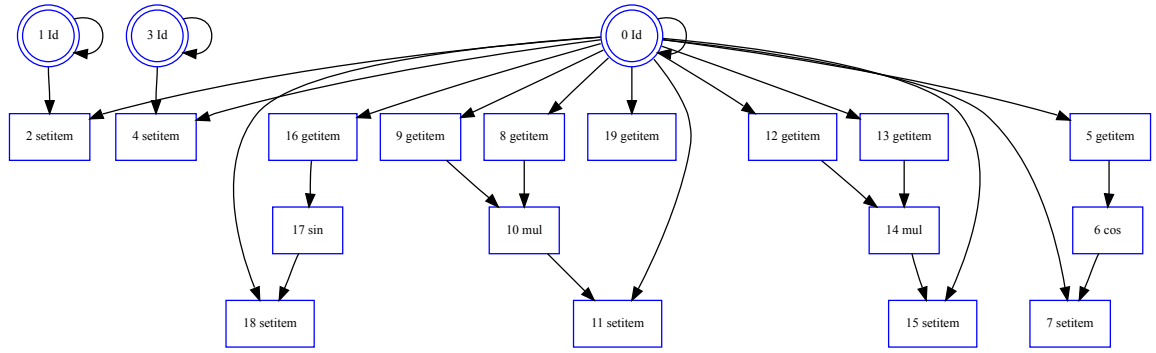
### 2.1.3. State Space Representation

There is also another useful representation of an algorithm that takes into account that a program is evaluated on some kind of register machine. The registers of the machine are memory locations in a buffer which contains the current *state* $s$ of the computation. One computational step overwrites (parts) of the state

$$s := \Phi_l(s) \ . \tag{2.2}$$

The functions $\Phi_l$, $l = 1, \ldots, L$ are called *elementary transitions*. Modern computers can perform billions of arithmetic operations per second. However, not all intermediate results are relevant and can often be discarded when the program evaluation advances. That means certain memory locations are overwritten and reused to store other temporary variables.

One advantage of the state space representation is that it allows one to write the complete




```
from numpy import array, sin, cos
from algopy import *
cg = CGraph()
x1 = 3; x2 = 7
x = Function(numpy.zeros(6))
x[0] = x1
x[1] = x2
x[2] = cos(x[1])
x[3] = x[2] * x[0]
x[4] = x[0] * x[3]
x[5] = sin(x[4])
y = x[5]
cg.plot('sideffectgraph.svg', method='dot')
```

Figure 2.2.: The computational graph of $y = \sin(x_1 + \cos(x_2) \cdot x_1)$ implemented using an array.

program as one composite function of the form

$$F = P_Y \circ \Phi_L \circ \cdots \circ \Phi_1 \circ P_X^T \,, \tag{2.3}$$

where $F : U \subseteq X \to Y$. The function $P_X^T$ maps the element $x \in U \subseteq X$ to the state space and $P_Y$ maps from the state space to $Y$. In other words

$$\begin{aligned} s^{(0)} =& P_X^T(x) \\ y =& P_Y(s^{(L)}) \,. \end{aligned}$$

Each elementary transition $\Phi_l$ can be described by

$$\Phi_l(s) := (\mathbf{1} - P_l P_l^T)s + P_l \phi_l(Q_l s) \,, \tag{2.4}$$

where the function $Q_l$ are the called *argument selections* that map the state space into the domains of the elementary functions. I.e., $v_{j \prec l} = Q_l(s)$ and $P_l$ the mapping of the result back to the state space.

**Example 2.1.2.** In state space representation, Example 2.1.1 results in the following single assignment procedure.

| step | state | operation | locations |
|---|---|---|---|
| -1 | $s = (3, 0, 0, 0, 0, 0)$ | assign variable | |
| 0 | $s = (3, 7, 0, 0, 0, 0)$ | assign variable | |
| 1 | $s = (3, 7, \cos(7), 0, 0, 0)$ | cos | $3 \leftarrow 2$ |
| 2 | $s = (3, 7, \cos(7), 3\cos(7), 0, 0)$ | multiplication | $4 \leftarrow 1, 3$ |
| 3 | $s = (3, 7, \cos(7), 3\cos(7), 3 + 3\cos(7), 0)$ | addition | $5 \leftarrow 1, 4$ |
| 4 | $s = (3, 7, \cos(7), 3\cos(7), 3 + 3\cos(7), \sin(3 + 3\cos(7))$ | sin | $6 \leftarrow 5$ |

The independent variables are $x = (3, 7)$ and the numerical value of the dependent variable $y = \sin(x_1 + \cos(x_2) \cdot x_1)$ can be found found after step 4 in the sixth location $s_6$ of the state

vector $s$. This is very similar to the internal representation of the computation in ADOL-C Griewank et al. [1999], see Table 2.1. Since the computation is performed in coordinates, it is possible to write the argument selections as matrices:

$$
\begin{aligned}
P_X^T x &= \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}^T \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \\
Q_1 s &= \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} s = s_2 \,, & P_1 \phi_1(Q_1 s) &= \begin{pmatrix} 0,0,1,0,0,0 \end{pmatrix}^T \phi_1(Q_1 s) \\
Q_2 s &= \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} s = \begin{pmatrix} s_1 \\ s_3 \end{pmatrix} \,, & P_2 \phi_2(Q_2 s) &= \begin{pmatrix} 0,0,0,1,0,0 \end{pmatrix}^T \phi_2(Q_2 s) \\
Q_3 s &= \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} s = \begin{pmatrix} s_1 \\ s_4 \end{pmatrix} \,, & P_3 \phi_3(Q_3 s) &= \begin{pmatrix} 0,0,0,0,1,0 \end{pmatrix}^T \phi_3(Q_3 s) \\
Q_4 s &= \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} s = s_5 \,, & P_4 \phi_4(Q_4 s) &= \begin{pmatrix} 0,0,0,0,0,1 \end{pmatrix}^T \phi_4(Q_4 s) \\
P_Y s &= \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} s = s_6 = y \,.
\end{aligned}
$$

**Example 2.1.3** (tracing with PyAdolc)**.** The inner workings of the algorithmic differentiation tool ADOL-C are based on the state space representation. To obtain a representation of the computation, the function evaluation is traced. I.e., the function evaluation is stored as a sequence of instructions and their arguments. Each variable is given a unique id. The Python bindings PyAdolc are used here to illustrate how ADOL-C works internally. All operations in Python are delegated to the underlying C++ calls of ADOL-C. The source code and the resulting trace is shown in Table 2.1. As one can see, the state space consists of at least 14 memory locations. Since the intermediate values of tmp1 is of temporary nature it is possible to reuse the memory location when tmp1 goes out of scope. ADOL-C performs checks to reduce the size of the state space. This is done when the assignment operator = is called. In Python it is not possible to overload the equality operator and therefore in PyAdolc it has been replaced by the operator <<=. The code is shown in Listing 2.2. It leads to the internal representation shown in Table 2.2. As one can see, only a state with 9 variables is required.

```
from numpy import sin, cos
from adolc import *

trace_on(1)
x1 = adouble(3.)
x2 = adouble(7.)
independent(x1)
independent(x2)
tmp1 = sin(x1 + cos(x2)*x1)
tmp2 = tmp1 * sin(x1)*cos(x2)
y = tmp1*tmp2
dependent(y)
trace_off()
```

Listing 2.1: Tracing with PyAdolc.

```
from numpy import sin, cos
from adolc import *

trace_on(2)
x1 = adouble(3.)
x2 = adouble(7.)
tmp1 = adouble(0.); tmp2 = adouble(0.)
independent(x1)
independent(x2)
tmp1 <<= sin(x1 + cos(x2)*x1)
tmp2 <<= tmp1 * sin(x1)*cos(x2)
y = tmp1*tmp2
```

| | | code | op | loc | loc | loc | loc | dbl | dbl | val | val | val | val |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 33 | start of tape | | | | | | | | | | | | |
| 1 | 16 | 4 | assign d | | | | 0 | | 3.00 | | | | 3.00 |
| 2 | 15 | 4 | assign d | | | | 1 | | 7.00 | | | | 7.00 |
| 3 | 14 | 1 | assign ind | | | | 0 | | 3.00 | | | | 3.00 |
| 4 | 13 | 1 | assign ind | | | | 1 | | 7.00 | | | | 7.00 |
| 5 | 12 | 20 | cos op | | 1 | 3 | 2 | | | | 7.00 | 0.66 | 0.75 |
| 6 | 11 | 15 | mult a a | | 2 | 0 | 3 | | | | 0.75 | 3.00 | 2.26 |
| 7 | 10 | 11 | plus a a | | 0 | 3 | 4 | | | | 3.00 | 2.26 | 5.26 |
| 8 | 9 | 21 | sin op | | 4 | 6 | 5 | | | | 5.26 | 0.52 | −0.85 |
| 9 | 8 | 21 | sin op | | 0 | 8 | 7 | | | | 3.00 | −0.99 | 0.14 |
| 10 | 7 | 15 | mult a a | | 5 | 7 | 9 | | | | −0.85 | 0.14 | −0.12 |
| 11 | 6 | 20 | cos op | | 1 | 11 | 10 | | | | 7.00 | 0.66 | 0.75 |
| 12 | 5 | 15 | mult a a | | 9 | 10 | 12 | | | | −0.12 | 0.75 | −0.09 |
| 13 | 4 | 15 | mult a a | | 5 | 12 | 13 | | | | −0.85 | −0.09 | 0.08 |
| 14 | 3 | 2 | assign dep | | | | 13 | | | | | | 0.08 |
| 15 | 2 | 0 | death not | | | 0 | 13 | | | | | 0.08 | −0.09 |
| 16 | 1 | 32 | end of tape | | | | | | | | | | |

Table 2.1.: This is the output of the code shown in Listing 2.1. One can see that in the tracing process, each variable is saved in a certain location loc of the state vector. As one can see, the state vector $s$ requires at least 14 locations since each intermediate result is given a unique id. The mult operation requires three locations: two for the input (e.g. above 2 and 0) and one for the output (3), whereas trigonometric function such as cos have one input (e.g. above 1) and two outputs (3 and 2). See Table 2.4 and Appendix A for an explanation. The following changes have been made to fit the table on one page.: double is abbreviated as as dbl and value as val.

```
dependent(y)
```

Listing 2.2: Tracing with PyAdolc with explicit value assignment. This mimics the C++ behavior where operator= is overloaded.

| | | code | op | loc | loc | loc | loc | dbl | dbl | val | val | val | val |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 33 | start of tape | | | | | | | | | | | | |
| 1 | 18 | 4 | assign d | | | | 0 | | 3.00 | | | | 3.00 |
| 2 | 17 | 4 | assign d | | | | 1 | | 7.00 | | | | 7.00 |
| 3 | 16 | 41 | assign d zero | | | | 2 | | | | | | 0.00 |
| 4 | 15 | 41 | assign d zero | | | | 3 | | | | | | 0.00 |
| 5 | 14 | 1 | assign ind | | | | 0 | | 3.00 | | | | 3.00 |
| 6 | 13 | 1 | assign ind | | | | 1 | | 7.00 | | | | 7.00 |
| 7 | 12 | 20 | cos op | | 1 | 5 | 4 | | | | 7.00 | 0.66 | 0.75 |
| 8 | 11 | 15 | mult a a | | 4 | 0 | 5 | | | | 0.75 | 3.00 | 2.26 |
| 9 | 10 | 11 | plus a a | | 0 | 5 | 6 | | | | 3.00 | 2.26 | 5.26 |
| 10 | 9 | 21 | sin op | | 6 | 8 | 2 | | | | 5.26 | 0.52 | −0.85 |
| 11 | 8 | 21 | sin op | | 0 | 5 | 4 | | | | 3.00 | −0.99 | 0.14 |
| 12 | 7 | 15 | mult a a | | 2 | 4 | 5 | | | | −0.85 | 0.14 | −0.12 |
| 13 | 6 | 20 | cos op | | 1 | 7 | 6 | | | | 7.00 | 0.66 | 0.75 |
| 14 | 5 | 15 | mult a a | | 5 | 6 | 3 | | | | −0.12 | 0.75 | −0.09 |
| 15 | 4 | 15 | mult a a | | 2 | 3 | 4 | | | | −0.85 | −0.09 | 0.08 |
| 16 | 3 | 2 | assign dep | | | | 4 | | | | | | 0.08 |
| 17 | 2 | 0 | death not | | | 0 | 8 | | | | | 0.08 | −0.09 |
| 18 | 1 | 32 | end of tape | | | | | | | | | | |

Table 2.2.: This is the output of the code shown in Listing 2.2. Compared to 2.1 a state space with only 9 locations is required.

## 2.2. Preliminaries and Notation

The purpose of this section is to collect well-known but fundamental results from calculus and numerical analysis that are important in subsequent sections. Furthermore, this section provides the notation and nomenclature.

**General Notation** The real numbers are denoted by $\mathbb{R}$ , the complex numbers by $\mathbb{C}$, $\mathbb{K}$ is some field, $\mathbb{N}$ are natural numbers, excluding zero, $\mathbb{N}_0$ the natural numbers including zero, $\mathbb{Z}$ the integer numbers. Superscripts of the form $\mathbb{R}^N$ denote the $N$-ary Cartesian product, i.e., $\mathbb{R}^N := \mathbb{R} \times \mathbb{R} \times \cdots \times \mathbb{R} = \{(x_1, \ldots, x_N) |\ x_n \in \mathbb{R} \text{ for all } n = 1, \ldots, N\}$. The symbols $X, Y, Z$ typically denote a general Banach space.

Elements of $\mathbb{R}^N$ are generally denoted by $u, v, w, x, y$ or $z$ and elements of $\mathbb{N}_0$ by $N, M, L, K$ and $i, j, n, m, l, k$. The big letters are mostly used to define a dimension or an upper bound of a summation and the small letters are used as indices. The notation $|S|$ denotes the number of elements in a finite set. To highlight (important) definitions or assignments $:=$ is used instead of $=$, e.g., $\mathbb{N}_0 := \mathbb{N} \cup \{0\}$. Furthermore, $:=$ is used to indicate that (a part of) a mutable data structure is overwritten.

**Definition 2.2.1** (sum, product, factorial, binomial coefficient)**.** *Sums* are denoted by

$$y = \sum_{n=1}^{N} x_n := x_1 + x_2 + \cdots + x_N \ ,$$

and *products* by

$$z = \prod_{n=1}^{N} x_n := x_1 \ldots x_N \ .$$

If $N < 1$ then $y = 0$ and $z = 1$, i.e., the identity elements of $+$ and $\cdot$. The *factorial* of an integer number $n \in \mathbb{Z}$ is defined by

$$n! := \begin{cases} \prod_{k=1}^{n} k & \text{if } n \geq 0, \\ 0 & \text{if } n < 0 \end{cases}$$

and the *binomial coefficient* by

$$\binom{x}{j} := \prod_{k=0}^{j-1} \frac{x-k}{j-k} \ ,$$

where $x \in \mathbb{R}$ and $j \in \mathbb{N}_0$. If $j = 0$ then the empty product returns 1.

**Definition 2.2.2** (multi-indices)**.** A *multi-index* is a tuple of natural numbers

$$\mathbf{i} = (i_1, i_2, \ldots, i_K) \in \mathbb{N}_0^K \ .$$

It is a generalization of an index and is used to simplify the notation of formulas. Let $\mathbf{i}, \mathbf{j} \in \mathbb{N}_0^K$ be to multi-indices and $x \in \mathbb{R}^K$.

$$|\mathbf{i}| := \sum_{k=1}^{K} i_k \ , \qquad x^{\mathbf{i}} := x_1^{i_1} \cdot \cdots \cdot x_K^{i_K} \ , \qquad \mathbf{i} \pm \mathbf{j} := (i_1 \pm j_1, \ldots, i_K \pm j_K)$$

$$\mathbf{i}! := \prod_{k=1}^{K} i_k! \ , \qquad \binom{\mathbf{i}}{\mathbf{j}} := \prod_{k=1}^{K} \binom{i_k}{j_k} \ , \qquad \mathbf{i} \leq \mathbf{j} \Leftrightarrow i_k \leq j_k \ , \ \forall k = 1, 2, \ldots$$

The set of all multi-indices with fixed sum is denoted

$$\mathbb{N}_0^K(d) := \{\mathbf{i} \in \mathbb{N}_0^K : |i| = d\} \ .$$

**Definition 2.2.3** (Slicing, Slice Indices)**.** A *slice index* is a generalization of an index in that it can be used to select more than one element at once. It is denoted by

$$i : j : k \ ,$$

where $i \in \mathbb{N}$ is the *lower bound*, $j \in \mathbb{N}$ the *upper bound* and $k \in \mathbb{N}$ the *step*. A *slice of a vector* $x$ is defined as follows:

$$x_{i:j:k} = [x_i, x_{i+k}, x_{i+2k}, \ldots, x_{i+lk}] \ ,$$

where $l \in \mathbb{N}$ the largest integer s.t. $i + lk \leq j$. The index $k$ in $i : j : k$ is optional. Omitting it is equivalent to $k = 1$ and the slice index is written as $i : j$. If $i$ is omitted it is is equivalent to $i = 1$, if $j$ is omitted then $j$ is the size of the list. One writes $: j$ in the first case and $i :$ in the second case. Consider for instance $x = [1, 3, 5, 7, 8, 13, 20]$. Then $x_{:3} = [1, 3, 5]$, $x_{2:5} = [3, 5, 7, 8]$, $x_{::2} = [1, 5, 8, 20]$.

**Memory Mapping of $K$-Tensors** The address space of the physical memory of a computer is linear, i.e., it can be thought of a one-dimenstional array. It is often necessary to define a function that maps from a mathematical object to an array, e.g., vectors and matrices. Let $A \in \mathbb{R}^{N_1 \times N_2 \times \cdots \times N_K}$ be a $K$-tensor with entries $A_{n_1,n_2,\ldots,n_K} \in \mathbb{R}$. The memory mapping is written as

$$\& : \mathbb{N}^K \to \mathbb{N}$$

and demand that it is bijective. A popular choice is the *row major* mapping

$$\mathbf{i} \mapsto \sum_{k=0}^{K-1} \prod_{j=1}^{k} N_{K-j+1}(i_{K-k} - 1) + 1 \ ,$$

where $\mathbf{i} \in \mathbb{N}^K$ is a multi-index and $(N_1, \ldots, N_K)$ are the upper bounds. For the special case of matrices, i.e., 2-tensors, one has $\mathbf{i} \mapsto N_2(i_1 - 1) + (i_2 - 1) + 1$. For example for $A \in \mathbb{R}^{2\times 3}$ one can access $(2, 1)$-th element at the 4-th array entry.

**Multi-Dimensional Arrays** A multi-dimensional array is described by a datatype, shape and stride. The datatype is a representation of a mathematical object, e.g., a real number as Float64 or an integer as Int64. The shape is a tuple of upper bounds, i.e.,$(N_1, \ldots, N_K)$. The number $K \in \mathbb{N}_0$ is also called the dimension or number of dimensions. The stride describes a particular choice of the memory mapping & and describes how many elements of the linear memory should be advanced when the $k$'th index is increased by one. The idea is most conveniently explained by an example. Consider the 3-tensor $A \in \mathbb{R}^{N_1 \times N_2 \times N_3}$ and let the stride be $s := (N_2N_3, N_3, 1)$. I.e., to go from $(i_1, i_2, i_3)$ to $(i_1 + 1, i_2, i_3)$ one has to advance $s_1 = N_3N_2$ entries. To get from $(i_1, i_2, i_3)$ to $(i_1, i_2 + 1, i_3)$ advance $N_3$ entries and from $(i_1, i_2, i_3)$ to $(i_1, i_2, i_3 + 1)$ one entry.

**Landau Symbols Notation** Let $f, g : \mathbb{R} \to \mathbb{R}$. One says

$$f(x) \in \mathcal{O}(g(x)) \text{ as } x \to \infty \ ,$$

(pronounced big-oh) if there exists some constant $M$ and a real number $x_0$ s.t.

$$\|f(x)\| \leq M\|g(x)\|$$

for all $x > x_0$. One says that $f$ is asymptotically bounded (up to a constant) above by $g$. Similarly on says $f(x) \in o(g(x))$ ($f(x)$ is little-o of $g(x)$) if

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = 0$$

More informally also the equality symbol $=$ is used, e.g., $f(x) = \mathcal{O}(x)$.

### 2.2.1. First-Order Derivatives

**Definition 2.2.4** (general notation)**.** Unless otherwise stated, $X = (X, \|\cdot\|)$ and $Y = (Y, \|\cdot\|)$ denote some Banach spaces over the same field. Open subsets of $X$ resp. $Y$ are written as $U \subseteq X$ and $V \subseteq Y$.

**Definition 2.2.5** (bounded linear operators, $\mathcal{L}(X, Y)$)**.** Let $X, Y$ be normed vector spaces and $A : X \to Y$ is linear. Then $A$ is called *bounded* if there exists $\alpha > 0$ s.t.

$$\|Ax\| \leq \alpha\|x\| \ , x \in X \ .$$

Define

$$\mathcal{L}(X, Y) := \{A \in \mathrm{Hom}(X, Y) \text{ and } A \text{ is bounded}\} \ ,$$

where $\mathrm{Hom}(X, Y)$ is the set of homomorphisms, i.e.,structure-preserving functions, which are the linear functions for vector spaces. Furthermore, let the set of bounded linear isomorphisms be denoted

$$\mathcal{L}is(X, Y) := \{A \in \mathcal{L}(X, Y) : A \text{ is an isomorphism}\} \ .$$

For notational convenience, the brackets of $A(x)$ are omitted for linear operators and thus resembles a matrix-vector product.

**Definition 2.2.6** (differentiability)**.** Let $F : U \subseteq X \to Y$ and $x \in U$, $U$ open subset of $X$. If there exists a function $A_x \in \mathcal{L}(X, Y)$, i.e.,linear and bounded, s.t.

$$F(x + v) = F(x) + A_x(v) + R(x; v)$$

for all $v$ s.t. $\|x + v\| \in U$ and

$$\lim_{v\to 0} \frac{R(x; v)}{\|v\|} = 0 \ ,$$

i.e.,$R(x; v) \in o(\|v\|)$, then the function $F$ is said to be *(Fréchet) differentiable at $x$.*

Equivalently, one can also define differentiability by $\lim_{v\to 0} \frac{f(x+v)-f(x)-A_x(v)}{\|v\|} = 0$. Of importance is the uniqueness of $A_x$ as guaranteed by the following

**Proposition 2.2.1.** *Let $F$ be differentiable at $x$. Then the linear operator $A_x \in \mathcal{L}(X, Y)$ is uniquely defined.*

*Proof.* See [Amann and Escher, 1999, chapter VII.2]. □

**Definition 2.2.7** ((total) derivative)**.** Let $F$ be defined as in the previous definition, i.e., differentiable at $x$. Then the uniquely defined operator $A_x$ is denoted

$$\begin{aligned} \partial F : U &\to \mathcal{L}(X, Y) \\ x &\mapsto \partial F(x; \cdot) \end{aligned}$$

and called the *(first) derivative of $F$ at $x$.* One writes

$$\partial F(x)v = \partial F(x;v) = A_x(v) \ .$$

**Definition 2.2.8** (continuous differentiability, $C^1(U)$)**.** Let $F : U \to Y$ be given. If $\partial F : U \mapsto \mathcal{L}(X,Y)$ is continuous, then the function $F$ is called continuously differentiable. One writes $F \in C^1$ or $F \in C^1(X,Y)$.

**Definition 2.2.9** (directional derivative)**.** Let $F : U \to Y$, $x \in U$ and $v \in X \setminus \{0\}$. There exists a sufficiently small $t$ such that $x + tv \in U$. If the function $t \mapsto F(x+tv)$ is differentiable at $t = 0$, then the derivative

$$\lim_{t\to 0} \frac{F(x+tv) - F(x)}{t}$$

is called the *directional derivative* or *Gâteaux derivative.*

**Proposition 2.2.2** (Chain Rule)**.** *Let $F : U \subseteq X \to Y$, $x \mapsto y = F(x)$ and $G : V \subseteq Y \to Z$, $y \mapsto z = G(y)$ be differentiable and $F(U) \subseteq V$, $X, Y, Z$ Banach spaces. Then $G \circ F : U \subseteq X \to Z$ is differentiable in $x$. The derivative is given as*

$$\partial(G \circ F)(x) = \partial G(y) \circ \partial F(x) \ .$$

*Proof.* According to the definition of differentiability it is possible to expand $F$ as follows

$$F(x+v) = F(x) + \partial F(x)v + R_x(x;v) \ .$$

Using this in the expansion of $G \circ F$ one obtains

$$\begin{aligned} G(F(x+v)) =& G(F(x) + \underbrace{\partial F(x)v + R_x(x;v)}_{=:w}) \\ =& (G \circ F)(x) + \partial G(F(x))w + R_y(y;w) \\ =& (G \circ F)(x) + \partial G(F(x))\partial F(x)v + \underbrace{\partial G(F(x))R_x(x;v) + R_y(y;w)}_{=:R(x;v)} \ . \end{aligned}$$

That means, if one can show that $\lim_{\|v\|\to 0} \frac{R(x;v)}{\|v\|} = 0$ then $\partial G \partial F = \partial(F \circ G)$ follows from the uniqueness of the derivative. One can check that

$$\begin{aligned} \lim_{\|v\|\to 0} \frac{R(x;v)}{\|v\|} =& \lim_{\|v\|\to 0} \frac{\partial G R_x(x;v)}{\|v\|} + \lim_{\|v\|\to 0} \frac{\partial R_y(y;w(v))}{\|v\|} \\ =& \partial G \lim_{\|v\|\to 0} \frac{R_x(x;v)}{\|v\|} + \lim_{\|w\|\to 0} \frac{\partial F C R_y(y;w)}{\|w\|} \\ =& 0 + \partial F C \lim_{\|w\|\to 0} \frac{R_y(y;w)}{\|w\|} = 0 \ , \end{aligned}$$

where the information that $\partial G$ and $\partial F$ are a bounded linear operators (and hence continuous) has been used, and $C > 0$ is a constant such that $\|w\| \leq C\|v\|$ for small enough $v$. □

### 2.2.2. Coordinate Representation

The above definitions are coordinate free: a property that is convenient from a theoretical point of view. However, for the simulation on a computer, explicit representations in coordinates are necessary. Of particular importance is the case $X = \mathbb{R}^N$, $Y = \mathbb{R}^M$ that allows one to write $G \in \mathcal{L}(X,Y)$ as matrices. The coordinate representation is written with enclosing square brackets []. I.e.,

$$[G] \in \mathbb{R}^{M\times N} \ .$$

Since $G(x) = G(\sum_{n=1}^{N} x_n e_n) = \sum_{n=1}^{N} G(e_n)x_n$ one can write $[G] = (G(e_1), \ldots, G(e_N))$. For $G_1, G_2$ linear functions one has the property $[G_1 \circ G_2] = [G_1] \cdot [G_2]$. The vectors $e_n \in \mathbb{R}^N$ are *Cartesian basis vectors*. To avoid clutter in the notation the brackets [] are omitted. It should be clear from the context when the coordinate representation is meant.

**Definition 2.2.10** (partial derivatives)**.** Let $F : U \subseteq \mathbb{R}^N \to \mathbb{R}^M$ and $x \in U$. Define the *partial derivative* of $F_m$ w.r.t. $x_n$ as

$$\mathrm{D}_n F_m(x) := \frac{\partial F_m}{\partial x_n}(x) := \lim_{t \to 0} \frac{F_m(x + te_n) - F_m(x)}{t} \in \mathbb{R}$$

when the limit exists.

**Definition 2.2.11** (Jacobian)**.** The *Jacobian* $J \in \mathbb{R}^{M \times N}$ is defined as the matrix of partial derivatives

$$(J(x))_{mn} = \mathrm{D}_n F_m = \frac{\partial F_m}{\partial x_n}$$

for $m = 1, \ldots, M$ and $n = 1, \ldots, N$ .

Fortunately, the next proposition states that under some conditions the Jacobian $J(x)$ and the derivative

$$F'(x) := \mathrm{D}F(x) = [\partial F(x)] \tag{2.5}$$

are described by the same matrix.

**Theorem 2.2.3.** *The function $F : U \in \mathbb{R}^N \to \mathbb{R}^M$ is continuously differentiable, i.e.,$F \in C^1$ if and only if $F$ is continuously partially differentiable. It then also holds that*

$$\partial F(x; v) = \mathrm{D}F(x)v \ ,$$

*where $\mathrm{D}F(x) \in \mathbb{R}^{M \times N}$.*

*Proof.* See Amann and Escher [1999]. □

Since it is typically clear from the context when the coordinate representation is used, the notational agreement

$$\partial F(x; v) \equiv \partial F(x)\{v\} \equiv \partial F(x)v = \mathrm{D}F(x)v \tag{2.6}$$

is made. In many cases one is interested in a linearization only in one argument of a function.

**Definition 2.2.12** (Notation of Functions with Several Arguments)**.** Let $F(y, t) \in \mathbb{R}^M$ be continuously differentiable in $y \in \mathbb{R}^N$ and $t \in \mathbb{R}$ with $y = x(t)$. The chain rule yields

$$\begin{aligned} \mathrm{D}F(y,t)\{\dot{y}, \dot{t}\} =& \mathrm{D}_y F(y,t)\dot{y} + \mathrm{D}_t F(y,t)\dot{t} \\ =& \mathrm{D}_y F(y,t)\mathrm{D}_t x(t)\dot{t} + \mathrm{D}_t F(y,t)\dot{t} \ . \end{aligned}$$

To avoid confusion when the function is written equivalently as $F(x(t), t)$, it is reasonable to introduce the following notation

$$\frac{\mathrm{d}F}{\mathrm{d}t}(x(t), t) = \frac{\partial F}{\partial y}(y, s)\bigg|_{y=x(t), s=t} \frac{\mathrm{d}x}{\mathrm{d}t}(t) + \frac{\partial F}{\partial s}(y, s)\bigg|_{y=x(t), s=t} \ ,$$

where $\frac{\partial F}{\partial y}(y, t) := \partial_y F(y, t)$. By $\frac{\partial F}{\partial s}(y, s)\Big|_{y=x(t), s=t}$ it is meant that after symbolic differentiation the symbol $y$ is substituted by $x(t)$ and the symbol $s$ by $t$.

### 2.2.3. Higher-Order Derivatives

If $g(x) := \partial F(x) \in \mathcal{L}(X, Y)$ is (continuously) differentiable, then $F$ is called twice (continuously) differentiable. The second derivative is denoted $\partial^2 F(x) \in \mathcal{L}(X, \mathcal{L}(X, Y))$. One can repeat this process to obtain higher-order derivatives. One can argue that once first-order derivatives are treated there is no need for an extra treatment of higher-order derivatives. However, one has to be careful not to neglect structure such as the symmetry of higher-order tensors; a property that should be taken into account.

**Definition 2.2.13** (bounded $m$-linear functions)**.** Let $X_1, \dots, X_m, Y$ be Banach spaces over the same field. A function $\phi : X_1 \times \cdots \times X_m \to Y$ is called *multi-linear* resp. *$m$-linear* if $\phi$ is linear in each of its variables. More explicitly, $\phi(\dots, x_n + cy_n, \dots) = \phi(\dots, x_n, \dots) + c\phi(\dots, y_n, \dots)$. Bounded $m$-linear functions are defined by

$$\mathcal{L}^m(X; Y) := \mathcal{L}(X_1, \dots, X_m; Y) \ .$$

**Proposition 2.2.4.** *There exists an isometric isomorphism from the space $\mathcal{L}(X_1, \dots, X_m)$ and $\mathcal{L}(X_1, \mathcal{L}(X_2, \dots, \mathcal{L}(X_m; Y))$ to $\mathcal{L}^m(X; Y)$.*

*Proof.* See Amann and Escher [1999]. □

**Definition 2.2.14** (continuous differentiability, $C^d(U)$)**.** Let $F : U \subseteq X \to Y$ be given. Define $\partial^0 F := F$ and assume that $\partial^{d-1} F : U \subseteq X \to \mathcal{L}(X, Y)$ exists. If

$$\partial^d F(x) := \partial(\partial^{d-1} F)(x) \in \mathcal{L}(X, \mathcal{L}^{d-1}(X, Y)) = \mathcal{L}^d(X, Y)$$

exists, then on calls $\partial^d F(x)$ the $d$-th derivative of $F$ in $X$. If additionally $\partial^d F(x)$ is continuous in $x$, then one says $F$ is $d$ times continuously differentiable and writes

$$F \in C^d(X, Y) \ .$$

This means in particular that $\partial^d F(x) \in \mathcal{L}(X_1, \dots, X_d; Y)$ is $d$-linear. The curly braces

$$\partial^d F(x)\{v_1, \dots, v_d\}$$

are used to indicate this linearity. In the case of linear maps the curly braces are often omitted, i.e.,$\partial F(x)\{v\} \equiv \partial F(x)v$.

**Proposition 2.2.5** (symmetry of derivative tensors)**.** *Let $f : X \to Y$ $d$-times continuously differentiable with $d \geq 2$. Then the derivative tensor is symmetric, i.e.,*

$$\partial^d f(x) \in \mathcal{L}^d_{sym}(X, Y) \ .$$

*Proof.* [Amann and Escher, 1999, Corollary 5.3] □

**Theorem 2.2.6** (Taylor's Theorem, univariate)**.** *Let $f \in C^d([-\epsilon, \epsilon])$ and $f \in C^{d+1}((-\epsilon, \epsilon))$, then*

$$f(t) = \sum_{k=0}^{d} \frac{1}{k!} \frac{\partial^k}{\partial t^k} f(t)\bigg|_{t=0} t^k + R_{d+1}(t) \ ,$$

*where the* remainder *in Lagrange form is given by*

$$R_{d+1}(f, t) := \frac{1}{(d+1)!} \frac{\partial^{d+1}}{\partial t^{d+1}} f(t)\bigg|_{t=\xi} t^d$$

*for some $\xi \in (-\epsilon, \epsilon)$. It is possible to estimate the remainder $R_{d+1}(f,t)$. If*

$$\left| \frac{\partial^{d+1}}{\partial t^{d+1}} f(t) \right| \leq M_{d+1} \quad \textit{for all } t \in (-\epsilon, \epsilon)$$

*then the remainder is bounded by*

$$|R_{d+1}(f,t)| \leq M_{d+1} \frac{\epsilon^{d+1}}{(d+1)!} .$$

**Theorem 2.2.7** (Taylor's Theorem, multivariate)**.** *Let $f : U \subset X \to Y$, $U$ open be a $d$-times continuously differentiable function. Then it holds that*

$$f(x+v) = \sum_{k=0}^{d} \frac{1}{k!} \partial^k f(x)\{v, \ldots, v\} + R_{d+1}(f,v) .$$

*In the special case $X = \mathbb{R}^N$ and $Y = \mathbb{R}$ this can be written in coordinates as*

$$f(x+v) = \sum_{|\mathbf{i}| \leq d} \frac{1}{\mathbf{i}!} \frac{\partial^{\mathbf{i}} f(x)}{\partial x^{\mathbf{i}}} v^{\mathbf{i}} + o(|v|^d) ,$$

*where $\mathbf{i} \in \mathbb{N}_0^N$ is a multi-index.*

### 2.2.4. Implicit Function Theorem

Of high importance is the implicit function theorem as it plays a central role in the differentiation of numerical linear algebra functions in Section 3.

**Proposition 2.2.8** (Implicit Function Theorem (IFT))**.** *Let $W$ be open in $X \times Y$, both Banach spaces, and $F \in C^d(W, Z)$. Furthermore $(x_0, y_0) \in W$ s.t.*

$$0 = F(x_0, y_0)$$

*and $\partial_y F(x,y)|_{(x,y)=(x_0,y_0)} \in \mathcal{L}is(Y, Z)$ . Then there exist open neighborhoods $U, V$ about $x_0$ and $(x_0, y_0)$ as well as a uniquely defined $G \in C^d(U, Y)$ with*

$$((x,y) \in V \textit{ and } F(x,y) = 0) \Leftrightarrow (x \in U \textit{ and } y = G(x)) .$$

*Proof.* See Amann and Escher [1999]. □

## 2.3. Univariate Taylor Polynomial Arithmetic

In this section it is explained in greater mathematical detail what is meant by "Univariate Taylor Polynomial" (UTP) arithmetic. Sufficiently smooth functions can be approximated locally by polynomials as shown in Figure 2.3. The basic idea is to introduce UTPs as a new *algebraic*

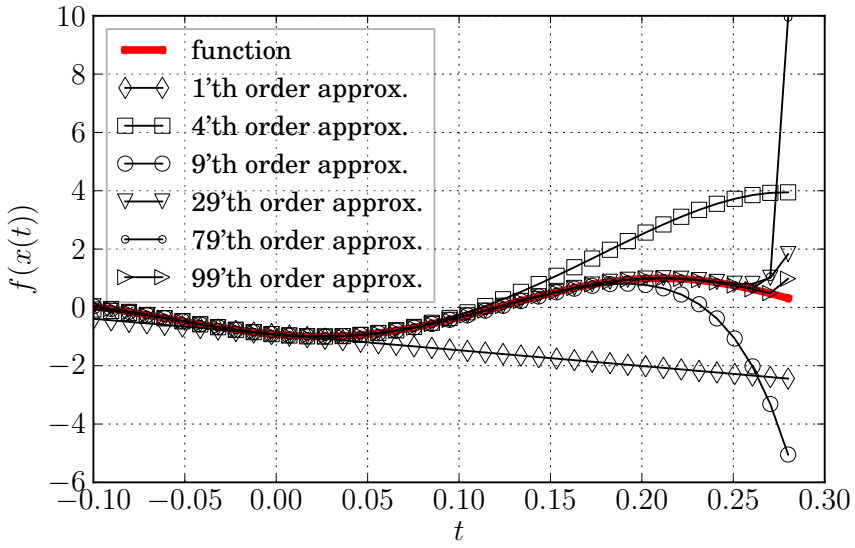



Figure 2.3.: Consider the function $f(x) = \sin(\log(\frac{\cosh(x_1)}{x_2} + x_3))$ and let $x(t) = (1,3,5)^T + (7,11,13)^T t$. The plot shows $f(x(t))$ evaluated at $t \in [-0.1, 0.28]$ and is labeled as "function". Additionally, Taylor series expansions or different orders $(1, 4, 9, 29, 79$ and $99)$ about $t = 0$ are shown. One can see that higher-order approximations approximate the function better in some interval $(-\epsilon, \epsilon)$ than lower order approximations. Also observe that higher-order expansions can explode when $\epsilon$ is chosen too large. This can be seen quite good at the 79th order approximation when $t \geq 0.23$.

*class* which is an extension of real arithmetic based on Taylor's theorem (see Chapter 2.2). An *order structure* is introduced in Chapter 2.3.2 to allow comparisons between UTPs. Also, a *topological structure* in briefly introduced in Chapter 2.3.3. The discussion is motivated from the work by Berz [1996] and Shamseddine [1999]. For the combination of the reverse mode of algorithmic differentiation and UTP arithmetic it is also useful to describe the operations in terms of Toeplitz matrices (Chapter 2.3.4). By use of the described topological structure one can describe what is meant by differentiation in UTP arithmetic (Chapter 2.3.5).

### 2.3.1. Algebraic Structure

Consider the sufficiently smooth function

$$\begin{aligned} F : \mathbb{R}^N &\to \mathbb{R}^M \\ x &\mapsto y = F(x) \ . \end{aligned}$$

For given curve $x(t) := \sum_{d=0}^{D-1} x_{[d]} t^d \in \mathbb{R}^N$ one would like to obtain

$$y(t) = F(x(t)) \ , \quad t \in (-\epsilon, \epsilon) \ .$$

Taylor's theorem states that

$$y(t) - \sum_{d=0}^{D-1} y_{[d]} t^d \in o(t^{D-1}) \ ,$$

where

$$y_{[d]} := \frac{1}{d!} \frac{\partial^d}{\partial t^d} x(t) \bigg|_{t=0} \ .$$

That means that the polynomial $\sum_{d=0}^{D-1} y_{[d]} t^d$ is a finite-dimensional approximation to the smooth curve $y(t)$. More precisely, the $\mathbb{R}^{D \times N}$ coefficients defining $x(t)$ are mapped to the $\mathbb{R}^{D \times M}$ coefficients describing an approximating polynomial of $y(t)$. It is useful to formalize this mapping between finite dimensional spaces in terms of the factor ring $\mathbb{R}[T]/(T^D)$. The reason for that is two-fold:

- The description as factor ring allows a generalization to more elaborate factor rings such as cross-derivatives
$$\mathbb{R}[T_1, \ldots, T_K]/(T_1^2, \ldots, T_K^2) \ ,$$
i.e., truncated polynomials with elements such as $T_1 T_3 T_4$ but not $T_1^2$ or $T_1^2 T_2^2$ or multivariate Taylor polynomials
$$\mathbb{R}[T_1, \ldots, T_K]/(T_1, \ldots, T_K)^D \ .$$

- Taylor's theorem involves concepts from calculus, i.e., taking limits of sequences. However, for the purpose of AD the mapping between Taylor coefficients has to be described by algebraic identities. It is therefore necessary to depart from the idea that $f(x(t))$ is a point-wise function for each possible realization of $x(t) \in \mathbb{R}$.

**Definition 2.3.1** (polynomial)**.** A polynomial over $\mathbb{R}$ is is a sequence $(x_{[d]})_{d \in \mathbb{N}_0}$ where only finitely many coefficients $x_{[d]} \in \mathbb{R}$ are nonzero, equipped with the two binary operators

$$\begin{aligned} (x_{[d]})_{d \in \mathbb{N}_0} + (y_{[d]})_{d \in \mathbb{N}_0} &= (x_{[d]} + y_{[d]})_{d \in \mathbb{N}_0} \\ (x_{[d]})_{d \in \mathbb{N}_0} \cdot (y_{[d]})_{d \in \mathbb{N}_0} &= (\sum_{i+j=d} x_{[i]} y_{[j]})_{d \in \mathbb{N}_0} \end{aligned}$$

One can also write the polynomial using the indeterminate $T$ as

$$[x]_\infty = \sum_{d=0}^{\infty} x_{[d]} T^d \in \mathbb{R}[T] \ . \tag{2.7}$$

The set of all polynomials is denoted

$$\mathbb{R}[T] := \left\{ [x]_\infty = \sum_{d=0}^{\infty} x_{[d]} T^d : x_{[d]} \in \mathbb{R} \text{ is nonzero only for finitely many } d \right\} \ .$$

*Remark.* The indeterminate $T$ plays a similar role as the imaginary number $i := \sqrt{-1}$ in the complex numbers $\mathbb{C}$. There is also some similarity in the binary operators $+$ and $\cdot$. In complex arithmetic, an element $x$ of the complex numbers is described by $x := \Re x + i \Im x$, where $\Re x \in \mathbb{R}$ is the real part and $\Im x \in \mathbb{R}$ the imaginary part of $x$. The complex multiplication $\mathbb{C} \times \mathbb{C} \to \mathbb{C}$ is defined by $xy := (\Re x + i \Im x)(\Re y + i \Re y) = (\Re x \Re y - \Im x \Im y) + i(\Re x \Im y + \Im x \Re y)$. As one can see, the imaginary part is computed in the same way as the first coefficient of a univariate Taylor polynomial. By choosing very small imaginary parts $\Im x$ and $\Im y$ one can make the expression $\Im x \Im y$ very small. This leads to the idea of the complex step derivative approximation described in more detail in Chapter 2.7.3.

**Definition 2.3.2** (supp, $\lambda$, $\overset{D}{=}$)**.** For $x, y \in \mathbb{R}[T]$ the following notations are used

$$\begin{aligned} \operatorname{supp}(x) &:= \{d \in \mathbb{N}_0 : x_{[d]} \neq 0\} \\ \lambda(x) &:= \min(\operatorname{supp}(x)) \ , \quad \lambda(0) := \infty \\ x &\overset{D}{=} y \qquad \text{if and only if } x_{[d]} = y_{[d]} \text{ for all } d = 0, \ldots, D-1 \ . \end{aligned}$$

**Definition 2.3.3** (degree)**.** The *degree* of a polynomial is defined by its largest nonzero coefficient, i.e., given $x \in \mathbb{R}[T]/(T^D)$, then the degree $\deg(x)$ is defined as

$$\deg(x) := \max\{d \in \mathbb{N}_0 : x_{[d]} \neq 0\} \ .$$

Typically, the symbol $d$ is used for the degree and also in many cases $D := d + 1$.

**Definition 2.3.4** (order)**.** The name *order* is typically used as synonym for degree. Additionally, order is used in the following sense: Let $x \in \mathbb{R}[T]$, then one says that $y = f(x) = 0$ is satisfied to order $D$ if $y_{[d]} = 0$ for all $d = 0, \ldots, D-1$ and $y_D \neq 0$.

**Definition 2.3.5** (general notation)**.** Elements of the polynomial ring $\mathbb{R}[T]$ of degree $D-1$ are written as

$$[x]_D := \sum_{d=0}^{D-1} x_{[d]} T^d \in \mathbb{R}[T] \ .$$

In addition, the following notation is used

$$\begin{aligned} [x]_D &:= [x_{[0]}, \ldots, x_{[D-1]}] := \sum_{d=0}^{D-1} x_{[d]} T^d \\ [x]_{D:D+E} &:= \sum_{d=D}^{D+E} x_{[d]} T^{d-D} \ . \end{aligned}$$

Compare this to the slicing of matrices (see Definition 2.2.3).

**Definition 2.3.6** (Factor Ring)**.** Consider the congruence relation $x \sim y$ if and only if $x - y \in T^D\mathbb{R}[T]$. It defines an equivalence class

$$\mathbb{R}[T]/(T^D) := \{x + (T^D) : x \in \mathbb{R}[T]\} \ ,$$

where

$$(T^D) := T^D\mathbb{R}[T] := \{xT^D : x \in \mathbb{R}[T]\}$$

defines a two-sided ideal. If the set $\mathbb{R}[T]/(T^D)$ is equipped with the arithmetic structure defined by the two binary operators

$$\begin{aligned} (x + (T^D)) + (y + (T^D)) &= (x + y) + (T^D) \\ (x + (T^D)) \cdot (y + (T^D)) &= (x \cdot y) + (T^D) \end{aligned}$$

it forms a ring. It is called *factor ring* or *quotient ring*. The representatives of minimal degree are denoted

$$[x]_D := \sum_{d=0}^{D-1} x_{[d]} T^d \in \mathbb{R}[T]$$

and are referred to as *representatives*. They are elements of the polynomial ring $\mathbb{R}[T]$. If not

stated otherwise, the binary operators + and · are functions from representative to representative. I.e.,

$$[z]_D = \sum_{d=0}^{D-1} z_{[d]} T^d := \sum_{d=0}^{D-1} \sum_{k=0}^{d} x_{[d-k]} y_{[k]} T^d \quad = \quad [x]_D \cdot [y]_D$$
$$[z]_D = \sum_{d=0}^{D-1} z_{[d]} T^d := \sum_{d=0}^{D-1} (x_{[d]} + y_{[d]}) T^d \quad = \quad [x]_D + [y]_D \ .$$

**Definition 2.3.7** (lifting and reducing)**.** Let $R$ be a ring and $I$ an ideal of $R$. Elements of $R/I$ can be represented by elements of $R$. Selecting an element $r \in R$ for a given element $s \in R/I$ is called *lifting* $R/I$ to $R$. Since the representative is not unique one may replace $r$ by $r - t$ for any $t \in I$. This is called *reducing* modulo $I$. See for instance Bernstein [2001] for the nomenclature.

**Definition 2.3.8.** Let $[x]_D \in \mathbb{R}[T]/(T^D)^N$ (note: we omit one set of brackets in the tuple $(\mathbb{R}[T]/(T^D))^N$ to improve readability), and let $F$ be $D$ times continuously differentiable. Define the operator

$$E_D : (\mathbb{R}^N \to \mathbb{R}^M) \to (\mathbb{R}[T]/(T^D)^N \to \mathbb{R}[T]/(T^D)^M)$$

by its action

$$\begin{aligned} E_D(F) : \mathbb{R}[T]/(T^D)^N &\to \mathbb{R}[T]/(T^D)^M \\ [x_1]_D, \ldots, [x_N]_D &\mapsto [y]_D = E_D(F)([x]_D) \\ E_D(F)([x_1]_D, \ldots, [x_N]_D) := \sum_{d=0}^{D-1} y_{[d]} T^d &= \sum_{d=0}^{D-1} \frac{1}{d!} \frac{\partial^d}{\partial T^d} F\Big(\sum_{d=0}^{D-1} x_{[d]} T^d\Big)\Big|_{T=0} T^d \ . \end{aligned}$$

The symbol $E_D$ is used to indicate that $E_D(F)$ *extends* functions that are defined on real numbers to functions on the factor ring $\mathbb{R}[T]/(T^D)$. Note that $T$ is one time considered to be an indeterminate and the other time to be real variable.

$$\begin{array}{ccccccc} T & \xrightarrow{x} & x(T) & \xrightarrow{f} & y(T) & \xrightarrow{g} & z(T) \\ \downarrow{\scriptstyle E_D(\mathrm{id})} & & \downarrow{\scriptstyle E_D(\mathrm{id})} & & \downarrow{\scriptstyle E_D(\mathrm{id})} & & \downarrow{\scriptstyle E_D(\mathrm{id})} \\ T & \xrightarrow{E_D(x)} & [x]_D & \xrightarrow{E_D(f)} & [y]_D & \xrightarrow{E_D(g)} & [z]_D \end{array}$$

Figure 2.4.: This commutative diagram illustrates that the Taylor series expansion of $g(f(x(T))) = (g \circ f \circ x)(T)$ is the same as the result of univariate Taylor polynomial arithmetic, i.e., $E_D(g \circ f \circ x) = E_D(g \circ f) \circ E_D(x)$. The function $x(\cdot)$ is given and $z(\cdot)$ is the desired quantity. From that point of view one computes $(g \circ f)(x(T))$. The function id is the identity $\mathrm{id}(x) = x$.

The definition of $E_D$ is compatible with the the usual definition of polynomials. This is useful since it provides a bridge between the calculus problem of computing Taylor polynomial approximations and algebraic computations on polynomials. This correspondence is now explained now in detail.

**Proposition 2.3.1.** *Let $f(x, y) = x + y$ and $g(x, y) = xy$ be the addition and multiplication of $x, y \in \mathbb{R}$. The extended operations*

$$E_D(f)([x]_D, [y]_D) \quad \textit{and} \quad E_D(g)([x]_D, [y]_D)$$

*are compatible with the addition and multiplication of the factor ring* $\mathbb{R}[T]/(T^D)$.

*Proof.* Check that the computed coefficients match those of Definition 2.3.6:

- addition: $E_D(f)([x]_D, [y]_D) = [x]_D + [y]_D$
- multiplication:

$$\begin{aligned} E_D(f)([x]_D, [y]_D) &= \sum_{d=0}^{D-1} \frac{1}{d!} \frac{\partial^d}{\partial T^d} \left( \sum_{i=0}^{D-1} x_{[i]} T^i \sum_{j=0}^{D-1} y_{[j]} T^j \right) \Bigg|_{T=0} T^d \\ &= \sum_{d=0}^{D-1} \sum_{k=0}^{d} x_{[d-k]} y_{[k]} T^d \ . \end{aligned}$$

□

**Lemma 2.3.2.** *Let* $f, g : \mathbb{R} \to \mathbb{R}$ *be* $D-1$ *times continuously differentiable, then the equations*

$$E_D(f \cdot g) = E_D(f) \cdot E_D(g)$$

*and*

$$E_D(f + g) = E_D(f) + E_D(g)$$

*hold.*

*Proof.* Let $y = f(x)$ and $z = g(x)$, then

$$\begin{aligned} E_D(f \cdot g)([x]_D) &= \sum_{d=0}^{D-1} \frac{1}{d!} \frac{\partial^d}{\partial T^d} f(x(T)) g(x(T)) \Bigg|_{T=0} T^d \\ &= \sum_{d=0}^{D-1} \frac{1}{d!} \sum_{k=0}^{d} \binom{d}{k} \left( \frac{\partial^{d-k}}{\partial T^{d-k}} f(x(T)) \right) \left( \frac{\partial^k}{\partial T^k} g(x(T)) \right) \Bigg|_{T=0} T^d \\ &= \sum_{d=0}^{D-1} \sum_{k=0}^{d} \frac{1}{d!} \frac{d!}{(d-k)!k!} \left( \frac{\partial^{d-k}}{\partial T^{d-k}} y(T) \right) \left( \frac{\partial^k}{\partial t^k} z(T) \right) \Bigg|_{T=0} T^d \\ &= \sum_{d=0}^{D-1} \sum_{k=0}^{d} \frac{1}{d!} \frac{d!}{(d-k)!k!} (d-k)! y_{[d-k]} k! z_{[k]} \Bigg|_{T=0} T^d \\ &= (E_D(f) \cdot E_D(g))([x]_D) \ . \end{aligned}$$

□

**Proposition 2.3.3.** *Let* $F : \mathbb{R}^N \to \mathbb{R}^M$ *and* $G : \mathbb{R}^M \to \mathbb{R}^K$ *be* $D-1$ *times continuously differentiable and. Then it holds*

$$E_D(G \circ F)([x]_D) = (E_D(G) \circ E_D(F))([x]_D) \ , \tag{2.8}$$

*i.e.,the operator* $E_D$ *is a homomorphism which preserves the function composition.*

*Proof.*

$$\begin{aligned}
E_D(G \circ F)([x]_D) &= \sum_{d=0}^{D-1} \frac{1}{d!} \frac{\partial^d}{\partial T^d} (G \circ F)(x(T)) \Big|_{T=0} T^d \\
&= \sum_{d=0}^{D-1} \frac{1}{d!} \frac{\partial^d}{\partial T^d} \left( G(\sum_{d=0}^{D-1} y_{[d]} T^d + r(T)) \right) \Big|_{T=0} T^d \\
&= \sum_{d=0}^{D-1} \frac{1}{d!} \frac{\partial^d}{\partial T^d} (G(\sum_{d=0}^{D-1} y_{[d]} T^d) \Big|_{T=0} T^d \\
&= \sum_{d=0}^{D-1} \frac{1}{d!} \frac{\partial^d}{\partial T^d} (G(E_D(F)([x]_D)) \Big|_{T=0} T^d \\
&= (E_D(G) \circ E_D(F))([x]_D) \ ,
\end{aligned}$$

where the definition of $E_D$ has been used in the first, second last and last row. From the second to the third line a first order Taylor expansion of $G$ in $\sum_{d=0}^{D-1} y_{[d]} T^d$ has been used, followed by the fact $r(T) \in o(T^{D-1})$. □

This basic result is the foundation of the forward mode of AD since it allows one to reduce any derivative computation of arbitrary functions to computations of elementary functions. The definition of the extended functions $E_D(f)$ is *compatible* with the algebraic structure of $\mathbb{R}[T]/(T^D)$. By that it is meant that the extension of the binary functions $+ : \mathbb{R}^2 \to \mathbb{R}$ and $\cdot : \mathbb{R}^2 \to \mathbb{R}$ yield the same recurrence.

### 2.3.2. Order Structure

In computer programs comparisons like $x \leq y$ and $x < y$ are ubiquituous. To be able to evaluate such programs in univariate Taylor polynomial arithmetic it is necessary to define what statements like $[x]_D \leq [y]_D$ mean.

**Definition 2.3.9** (partial order)**.** Let $x$ be element of some set $X$ and let $\leq$ be a binary relation. If $\leq$ satisfies the statements

1. If $x \leq y$ and $y \leq x$ then $y = x$ (*antisymmetry*);
2. If $x \leq y$ and $y \leq z$ then $x \leq z$ (*transitivity*);
3. $x \leq x$ (*reflexivity*);

it is called *partial order*. The set together with the relation $\leq$ is called *partially ordered set*.

**Definition 2.3.10** (comparable)**.** Let $x \neq y \in X$. If $x \leq y$ or $y \leq x$ one says that $x$ and $y$ are *comparable*. Otherwise they are *incomparable*.

**Definition 2.3.11** (total order)**.** Let $X$ be some set and $\leq$ be a binary relation. If $\leq$ satisfies the statements

1. If $x \leq y$ and $y \leq x$ then $y = x$ (*antisymmetry*);
2. If $x \leq y$ and $y \leq z$ then $x \leq z$ (*transitivity*);
3. $x \leq y$ or $y \leq x$ (*totality*);

it is called *total order*. The set together with the relation $\leq$ is called a *totally ordered set*.

I.e., a totally ordered set means that any two distinct elements of a partially ordered set are comparable.

**Definition 2.3.12** (strict total order)**.** Let $x, y \in X$, where $X$ is a totally ordered set w.r.t. $\leq$. One says $x$ is less than $y$ if and only if $x \neq y$ and $x \leq y$. This relation is denoted $x < y$.

**Lemma 2.3.4.** *Let $[x]_D, [y]_D \in \mathbb{R}[T]/(T^D)$ with $[x]_D \neq [y]_D$. Then the relation*

$$[x]_D \leq [y]_D \quad \textit{if and only if} \quad x_{[0]} \leq y_{[0]}$$

*does not define a partial order.*

*Proof.* By counterexample: Let $[x]_2 = 0+T$ and $[y]_2 = 0+0T$. Then $[x]_2 \leq [y]_2 = 0 \leq 0 = \text{True}$ and $[y]_2 \leq [x]_2 = 0 \leq 0 = \text{True}$, however $[x]_2 \neq [y]_2$. Therefore the antisymmetry is violated. □

**Definition 2.3.13** (lexicographical order of univariate Taylor polynomials)**.** Let $[x]_D, [y]_D \in \mathbb{R}[T]/(T^D)$. One says $[x]_D$ is less equal $[y]_D$, in formulas $[x]_D \leq [y]_D$, if and only if for $k = \lambda([x]_D - [y]_D)$

$$x_{[k]} \leq y_{[k]} \ .$$

If $k = \infty$ then $x_{[k]} = y_{[k]} = 0$ and hence $x_{[k]} \leq y_{[k]}$ holds.

**Definition 2.3.14** (set of positive univariate Taylor polynomials)**.** Define the set of positive univariate Taylor polynomials $\mathbb{R}[T]^+/(T^D)$ as

$$\mathbb{R}^+[T]/(T^D) := \{[x]_D \in \mathbb{R}[T]/(T^D) : [x]_D > 0\} \ .$$

The definition depends on the choice of the order relation $\leq$.

### 2.3.3. Topological Structure

Two topologies are introduced: one order-induced topology and one norm-induced topology. Later on, only the norm-induced topology will be used. The idea is motivated from the discussion in Berz [1996], Shamseddine [1999] who investigated the properties of the Levi-Civita field. According to them it is advantageous to make a distinction between the absolute value $|[x]_D|$ which is defined based on an order relation and the norm $\|[x]_D\|$.

**Definition 2.3.15** (absolute value)**.** Let $[x]_D \in \mathbb{R}[T]/(T^D)$, then the *absolute value* $|[x]_D| \in \mathbb{R}^+[T]/(T^D)$ of $[x]_D$ is defined as

$$|[x]_D| := \begin{cases} [x]_D & \text{if } [x]_D \geq 0 \\ -[x]_D & \text{otherwise} \end{cases}$$

**Definition 2.3.16** (Order Topology)**.** Call a subset $S$ of $\mathbb{R}[T]/(T^D)$ open if and only if for any $[y]_D \in S$ there exists $[\epsilon]_D > 0 \in \mathbb{R}[T]/(T^D)$ such that the open ball

$$B([y]_D, [\epsilon]_D) := \{[x]_D : |[x]_D - [y]_D| < [\epsilon]_D\}$$

is a subset of $S$.

**Definition 2.3.17** (Vector Mapping)**.** A bijective function vec from the UTPs to a real vector space defined by

$$\begin{aligned} \text{vec} : \mathbb{R}[T]/(T^D) &\to \mathbb{R}^D \\ [x]_D &\mapsto (x_{[0]}, \ldots, x_{[D-1]})^T \end{aligned}$$

is called the *vector mapping*.

Since any $[x]_D$ can be viewed as an element of $\mathbb{R}^D$ the norm can be induced from the vector space's norm.

**Definition 2.3.18** (Norm of UTPs)**.** Define the norm of a univariate Taylor polynomial as

$$\|[x]_D\| := \|\mathrm{vec}([x]_D)\| \ .$$

**Definition 2.3.19** (Norm Topology)**.** The subset $S$ of $\mathbb{R}[T]/(T^D)$ is called *open* if and only if for any $[y]_D \in S$ there exists $\epsilon > 0 \in \mathbb{R}$ such that the open ball

$$B([y]_D, \epsilon) := \{[x]_D : \|[x]_D - [y]_D\| < \epsilon\}$$

is a subset of $S$.

### 2.3.4. Isomorphism to Toeplitz-Matrix Calculus

There is a ring-isomorphism between the polynomial factor ring $\mathbb{R}[T]/(T^D)$ and the Toeplitz-matrices. I.e., one maps the $D$ coefficients $[x]_D \in \mathbb{R}^[T]/(T^D)$ to a matrix with Toeplitz-structure

$$[x]_D \mapsto \begin{pmatrix} x_{[0]} & & & & \\ x_{[1]} & x_{[0]} & & & \\ x_{[2]} & x_{[1]} & x_{[0]} & & \\ \vdots & \ddots & \ddots & \ddots & \\ x_{[D-1]} & \dots & x_{[2]} & x_{[1]} & x_{[0]} \end{pmatrix} . \tag{2.9}$$

**Proposition 2.3.5** (ring-isomorphism between univariate Taylor polynomials and Toeplitz–matrices)**.** *Let $[x]_D \in \mathbb{R}[T]/(T^D)$, then the mapping*

$$\begin{aligned} \mathrm{toep} : \mathbb{R}[T]/(T^D) &\to \mathbb{R}^{D\times D} \\ [x]_D &\mapsto \mathrm{toep}([x]_D) = (x_{[i-j]}\delta_{i\geq j})_{i,j=0,\dots,D-1} \end{aligned}$$

*is a ring-isomorphism. I.e.,*

1. $\mathrm{toep}([x]_D + [y]_D) = \mathrm{toep}([x]_D) + \mathrm{toep}([y]_D)$
2. $\mathrm{toep}([x]_D[y]_D) = \mathrm{toep}([x]_D)\mathrm{toep}([y]_D)$
3. $\mathrm{toep}([1, 0, \dots, 0]) = \mathbf{I}_D$ .

*Proof.* The first point and third point are clear. For convenience it is counted from zero and hence the top left element of a matrix $A$ is $A_{00}$. Let $[z]_D = [x]_D[y]_D$. Then the matrix element $\mathrm{toep}([z]_D)_{ij}$ is defined by $z_{[i-j]}\delta_{i\geq j}$ and is computed by the Cauchy-product as $\sum_{k=0}^{i-j} x_{[i-j-k]}y_{[k]}$. On the other hand,

$$\begin{aligned} (\mathrm{toep}([x]_D)\mathrm{toep}([y]_D))_{ij} &= \sum_{k=0}^{D-1} x_{ik}y_{kj} = \sum_{k=0}^{D-1} x_{i-j}\delta_{i\geq k}y_{k-j}\delta_{k\geq j} \\ &= \sum_{k=j}^{i} x_{i-k}y_{k-j} = \sum_{l=0}^{i-j} x_{i-j-l}y_l \ . \end{aligned}$$

□

*Remark.* Vectors and matrices can be written as block-Toeplitz matrices. Let $[A]_D \in \mathbb{R}[T]/(T^D)^{N\times M}$, then

$$\operatorname{toep}([A]_D) = \begin{pmatrix} A_{[0]} & & & & \\ A_{[1]} & A_{[0]} & & & \\ A_{[2]} & A_{[1]} & A_{[0]} & & \\ \vdots & \dots & \ddots & \ddots & \\ A_{[D-1]} & \dots & A_{[2]} & A_{[1]} & A_{[0]} \end{pmatrix},$$

where $A_{[d]} \in \mathbb{R}^{N\times M}$ for $d = 0, \dots, D-1$.

*Remark.* One can write the polynomial multiplication $[z]_D = [x]_D[y]_D$ also as Toeplitz $\times$ vector product

$$\begin{pmatrix} z_{[0]} \\ z_{[1]} \\ \vdots \\ z_{[D-1]} \end{pmatrix} = \begin{pmatrix} x_{[0]} & & & \\ x_{[1]} & x_{[0]} & & \\ \vdots & \ddots & \ddots & \\ x_{[D-1]} & \dots & & x_{[0]} \end{pmatrix} \begin{pmatrix} y_{[0]} \\ y_{[1]} \\ \vdots \\ y_{[D-1]} \end{pmatrix}. \tag{2.10}$$

### 2.3.5. Differentiability

Since the polynomials can be endowed with a topology and can be regarded as finite dimensional real vector space it is possible to ask for the Fréchet derivative. Basically, one computes the Jacobian of the mapping $(x_{[0]}, \dots, x_{[D-1]}) \mapsto (y_{[0]}, \dots, y_{[D-1]})$. One finds that the directional derivative in direction $[\dot{x}]_D$ can be written as a polynomial multiplication. Or put differently: the Jacobian takes a block-Toeplitz form.

**Proposition 2.3.6.** *Let $F : \mathbb{R}^N \to \mathbb{R}^M$, $x \mapsto y = F(x)$ be $D$ times continuously differentiable in some neighborhood of $x_{[0]}$ and let*

$$\begin{aligned} E_D(F) : \mathbb{R}[T]/(T^D)^N &\to \mathbb{R}[T]/(T^D)^M \\ [x]_D &\mapsto [y]_D = E_D(F)([x]_D) \end{aligned}$$

*be its extended function. Then it holds that each $y_{[d]}$ is a once continuously differentiable function w.r.t. $x_{[0]}, \dots, x_{[d]}$ and*

$$\frac{\partial y_{[i]}}{\partial x_{[j]}} = \begin{cases} J_{[i-j]} & \text{if } i \geq j \\ 0 & \text{else} \end{cases}, \tag{2.11}$$

*where the Jacobian coefficients $J_{[d]} \in \mathbb{R}^{M\times N}$ are*

$$[J]_D := E_D(\partial F)([x]_D). \tag{2.12}$$

*Proof.* As described by Christianson [1991] and Griewank and Walther [2008] this follows from

the straight-forward calculation

$$\begin{aligned}
y_{[i]} =& \frac{1}{i!}\frac{\partial^i}{\partial T^i} F(\sum_{d=0}^{D-1} x_{[d]} T^d)\Big|_{T=0} \\
\text{therefore} \quad \frac{\partial y_{[i]}}{\partial x_{[j]}} =& \frac{1}{i!}\frac{\partial^i}{\partial T^i}\frac{\partial}{\partial x_{[j]}} F(\sum_{d=0}^{D-1} x_{[d]} T^d)\Big|_{T=0} \\
=& \frac{1}{i!}\frac{\partial^i}{\partial T^i} F'(\sum_{d=0}^{D-1} x_{[d]} T^d) T^j \delta_{j<D}\Big|_{T=0} \\
=& \frac{1}{i!}\sum_{k=0}^{i}\binom{i}{k}\left(\frac{\partial^{i-k}}{\partial T^{i-k}} F'(\sum_{d=0}^{D-1} x_{[d]} T^d)\right)\Big|_{T=0} j!\delta_{kj}\delta_{i\geq j} \\
=& \frac{j!}{i!}\frac{i!}{(i-j)!j!}\frac{\partial^{i-j}}{\partial T^{i-j}} F'(\sum_{d=0}^{D-1} x_{[d]} T^d)\Big|_{T=0} \delta_{i\geq j} \\
=& J_{[i-j]}\delta_{i\geq j} \ ,
\end{aligned}$$

where Leibniz's rule and the interchangeability of the derivatives is used. □

**Corollary 2.3.7.** *The product*

$$[\dot{y}]_D = E_D(\partial F)([x]_D)[\dot{x}]_D$$

*reads in Toeplitz-form*

$$\begin{pmatrix} \dot{y}_{[0]} & & & \\ \dot{y}_{[1]} & \dot{y}_{[0]} & & \\ \vdots & \ddots & \ddots & \\ \dot{y}_{[D-1]} & \cdots & \dot{y}_{[1]} & \dot{y}_{[0]} \end{pmatrix} = \begin{pmatrix} J_{[0]} & & & \\ J_{[1]} & J_{[0]} & & \\ \vdots & \ddots & \ddots & \\ J_{[D-1]} & \cdots & J_{[1]} & J_{[0]} \end{pmatrix} \begin{pmatrix} \dot{x}_{[0]} & & & \\ \dot{x}_{[1]} & \dot{x}_{[0]} & & \\ \vdots & \ddots & \ddots & \\ \dot{x}_{[D-1]} & \cdots & \dot{x}_{[1]} & \dot{x}_{[0]} \end{pmatrix}. \tag{2.13}$$

**Corollary 2.3.8** (differentiability of extended functions)**.** *Let $F : \mathbb{R}^N \to \mathbb{R}^M$ be $D$ times continuously differentiable and let $E_D(F) : \mathbb{R}[T]/(T^D)^N \to \mathbb{R}[T]/(T^D)^M$ its extended function. Then $E_D(F)$ is once continuously differentiable and*

$$\partial(E_D(F))([x]_D)\{[h]_D\} = E_D(\partial F)([x]_D)\cdot[h]_D \ .$$

*Proof.* Norms are defined as in Definition 2.3.18. Since the derivative is unique it remains to show that $E_D(\partial F)([x]_D)\cdot[h]_D$ satisfies the requirements, i.e.

$$0 = \lim_{\|[h]_D\|\to 0} \frac{\|E_D(F)([x]_D+[h]_D) - E_D(F)([x]_D) - E_D(\partial F)([x]_D)\cdot[h]_D\|}{\|[h]_D\|} \ .$$

To see that, regard $E_D(F)$ as a mapping from the $D$ coefficients $x_{[0]},\ldots,x_{[D-1]}$ to $D$ coefficients $y_{[0]},\ldots,y_{[D-1]}$. In the previous proposition it has been shown that each $y_{[d]}$ is once continuously

differentiable in $x_{[0]}, \ldots, x_{[d]}$ and hence

$$
\begin{aligned}
E_D(F)([x]_D + [h]_D) &= \begin{pmatrix} F_{[0]}(x_{[0]} + h_{[0]}) \\ F_{[1]}(x_{[0]} + h_{[0]}, x_{[1]} + h_{[1]}) \\ \vdots \\ F_{[D-1]}(x_{[0]} + h_{[0]}, x_{[1]} + h_{[1]}, \ldots, x_{[D-1]} + h_{[D-1]}) \end{pmatrix} \\
&= \begin{pmatrix} F_{[0]}(x_{[0]}) + \partial F_{[0]}(x_{[0]})h_{[0]} + o(h_{[0]}) \\ F_{[1]}(x_{[0]}, x_{[1]}) + \partial_{x_{[0]}} F_{[1]}(x_{[0]}, x_{[1]})h_{[0]} + o(h_{[0]}) + \partial_{x_{[1]}} F_{[1]}(x_{[0]}, x_{[1]})h_{[1]} + o(h_{[1]}) \\ \vdots \\ F_{[D-1]}(x_{[0]}) + \sum_{k=0}^{D-1} J_{[D-1-k]} x_{[k]} + o(h_{[k]}) \end{pmatrix} \\
&= E_D(F)([x]_D) + [J]_D[h]_D + o([h]_D) \, .
\end{aligned}
$$

Inserting this in the requirements yields the wanted result. □

Regarding the interpretation of the entries $J_{[d]}, d = 0, \ldots, D-1$ as derivatives one uses the chain rule and obtains [Griewank and Walther, 2008, chapter 13.2]

$$
\begin{aligned}
J_{[0]} &= \partial F(x_{[0]}) \\
J_{[1]} &= \frac{1}{1!} \frac{\partial}{\partial T} \partial F(\sum_{d=0}^{D-1} x_{[d]} T^d)\Big|_{T=0} = \partial^2 F(x_{[0]}) x_{[1]} \\
J_{[2]} &= \frac{1}{2!} \frac{\partial^2}{\partial T^2} \partial F(\sum_{d=0}^{D-1} x_{[d]} T^d)\Big|_{T=0} = \partial^2 F(x_{[0]}) x_{[2]} + \frac{1}{2} \partial^3 F(x_{[0]})\{x_{[1]}, x_{[1]}\} \\
J_{[3]} &= \frac{1}{3!} \frac{\partial^3}{\partial T^3} \partial F(\sum_{d=0}^{D-1} x_{[d]} T^d)\Big|_{T=0} = \partial^2 F(x_{[0]}) x_{[3]} + \partial^3 F(x_{[0]})\{x_{[1]}, x_{[2]}\} + \frac{1}{6} \partial^4 F(x_{[0]})\{x_{[1]}, x_{[1]}, x_{[1]}\}.
\end{aligned}
$$

As one can see, a clever choice of the input coefficients yields higher-order directional derivatives.

### 2.3.6. Algorithms for Elementary Functions

Proposition 2.3.3 states that Taylor series expansions of arbitrary complex computer programs can easily be computed by $L$ successive pushforwards of Taylor polynomials through elementary functions $\phi_l$, $l = 1, \ldots, L$. The obvious next step is to provide algorithms for $E_D(\cdot)$, $E_D(\div)$, $E_D(\exp)$, etc. Before discussing how such algorithms can be derived, it is instructive to ask the question:

*What is an elementary function?*

In principle, only the definition of addition and multiplication is required for the definition of an algebraic class. Other functions are either implicitly defined or composite functions. This is a fundamental requirement which also allows one to evaluate floating point approximations of mathematical functions on a CPU. For instance, the exponential function can be computed by truncating the infinite summation

$$
y = \exp(x) = \sum_{d=0}^{\infty} \frac{1}{d!} x^d
$$

at some $d = K$. However, while theoretically possible, it is not a good idea in practice to regard functions such as $\exp(x)$ as sequence of additions and multiplications. The reason is that one would like to reuse existing and well-tested code such as available in the math libraries that come with a programming language. Such existing algorithms use many tricks and may require

program branches that are likely to be questionable from the differentiability point of view. Also, the algorithms are typically tuned in a way to allow fast execution times on a computer. Replacing all multiplications and additions in these algorithms with extended versions of the multiplication and addition is likely to introduce additional overhead. Therefore it is a good idea to derive structure exploiting algorithms that compute $E_D(\phi)$ for the elementary functions $\phi \in \{\pm, \cdot, /, \sin, \cos, \dots\}$ based on their mathematical definition. For convenience all functions enlisted in the C header file *math.h* are called *basic elementary functions* in the following.

There are two classes of basic elementary functions:

1. functions defined by algebraic equations
2. functions defined by differential equations.

An example of the first kind is the division $z = \frac{x}{y} = \div(x, y)$. One would like to know how to compute the first $D$ coefficients $z_{[d]}$, $d = 0, \dots, D-1$ from $[x]_D$ and $[y]_D$. The division is defined as the solution of $0 = zy - x$. In Taylor polynomial arithmetic this means that $0 = [x]_D - [z]_D[y]_D \mod T^D$ has to be satisfied. By comparing coefficients one finds for $d = 0$ that $0 = x_{[0]} - z_{[0]}y_{[0]}$, for $d = 1$ that $0 = x_{[1]} - (z_{[0]}y_{[1]} + z_{[1]} + y_{[0]})$ and in general

$$0 = x_{[d]} - \sum_{k=0}^{d} z_{[k]}y_{[d-k]} = x_{[d]} - \sum_{k=0}^{d-1} z_{[k]}y_{[d-k]} - z_{[d]}y_{[0]}$$

for $d = 0, \dots, D-1$. One hence obtains the recurrence

$$\begin{aligned} z_{[0]} &= \frac{x_{[0]}}{y_{[0]}} \\ z_{[d]} &= \frac{1}{y_{[0]}} \left( x_{[d]} - \sum_{k=0}^{d-1} z_{[k]}y_{[d-k]} \right) . \end{aligned}$$

This derivation is close to the approach taken in Chapter 3.

As an example of the second kind consider again the exponential function. It can be defined as solution of the ordinary differential equation $\frac{\partial y}{\partial x} = y, y(0) = 1$. Using the chain rule one obtains $\frac{\mathrm{d}y(x(t))}{\mathrm{d}t} = \frac{\partial y}{\partial x}\frac{\mathrm{d}x}{\mathrm{d}t}$ and thus

$$y(t)\dot{x}(t) = \dot{y}(t) .$$

That means one can compute the unknown coefficients in $y(t) = \sum_{d=0}^{\infty} y_{[d]}t^d$ from the known coefficients $x(t) = \sum_{d=0}^{\infty} x_{[d]}t^d$.

In the Tables 2.3 and 2.4 one can find a collection of the most important elementary functions. Hyperbolic and inverse trigonometric functions can be found in Appendix A.

| $z = \phi(x, y)$ | $d = 0, \dots, D$ | OPS: $\cdot$ | $\pm$ | nl | OPS | MV |
|---|---|---|---|---|---|---|
| $x + cy$ | $z_{[d]} = x_{[d]} + cy_{[d]}$ | $D$ | $D$ | 0 | $\sim 2D$ | $3D$ |
| $x \cdot y$ | $z_{[d]} = \sum_{k=0}^{d} x_{[k]}y_{[d-k]}$ | $\frac{(D+2)(D+1)}{2}$ | $\frac{(D+1)D}{2}$ | 0 | $\sim D^2$ | $3D$ |
| $x/y$ | $z_{[d]} = \frac{1}{y_0}\left[x_{[d]} - \sum_{k=0}^{d-1} z_{[k]}y_{[d-k]}\right]$ | $\frac{(D+1)D}{2}$ | $\frac{D(D-1)}{2} + 1$ | 1 | $\sim D^2$ | $3D$ |

Table 2.3.: Taylor polynomial algorithms for the binary operators $\pm, \cdot, \div$. OPS is the number of arithmetic operations, grouped into multiplication $\cdot$, addition/subtraction $\pm$ and nonlinear operations nl. MV is the number of elements which have to be moved from the main memory to the registers of a CPU and back.

| $y = \phi(x)$ | $d = 1, \ldots, D$ | OPS | MOVES |
|---|---|---|---|
| $\ln(x)$ | $\tilde{y}_{[d]} = \frac{1}{x_{[0]}} \left[ \tilde{x}_{[d]} - \sum_{k=1}^{d-1} x_{[d-k]} \tilde{y}_{[k]} \right]$ | $\sim D^2$ | $2D$ |
| $\exp(x)$ | $\tilde{y}_{[d]} = \sum_{k=1}^{d} y_{[d-k]} \tilde{x}_{[k]}$ | $\sim D^2$ | $2D$ |
| $\sqrt{x}$ | $y_{[d]} = \frac{1}{2y_{[0]}} \left[ x_{[d]} - \sum_{k=1}^{d-1} y_{[k]} y_{[d-k]} \right]$ | $\sim \frac{1}{2} D^2$ | $2D$ |
| $x^r$ | $\tilde{y}_{[d]} = \frac{1}{x_{[0]}} \left[ r \sum_{k=1}^{d} y_{[d-k]} \tilde{x}_{[k]} - \sum_{k=1}^{d-1} x_{[d-k]} \tilde{y}_{[k]} \right]$ | $\sim 2D^2$ | $2D$ |
| $\sin(x)$<br>$\cos(x)$ | $\tilde{s}_{[d]} = \sum_{k=1}^{d} \tilde{x}_{[k]} c_{[d-k]}$<br>$\tilde{c}_{[d]} = \sum_{k=1}^{d} -\tilde{x}_{[k]} s_{[d-k]}$ | $\sim 2D^2$ | $3D$ |
| $\tan(x)$ | $\tilde{y}_{[d]} = \sum_{k=1}^{d} w_{[d-k]} \tilde{x}_{[k]}$<br>$\tilde{w}_{[d]} = 2 \sum_{k=1}^{d} y_{[d-k]} \tilde{y}_{[k]}$ | $\sim 2D^2$ | $2D$ |
| $\arcsin(x)$ | $\tilde{y}_{[d]} = w_{[0]}^{-1} \left( \tilde{x}_{[d]} - \sum_{k=1}^{d-1} w_{[d-k]} \tilde{y}_{[k]} \right)$<br>$\tilde{w}_{[d]} = -\sum_{k=1}^{d} x_{[d-k]} \tilde{y}_{[k]}$ | $\sim 2D^2$ | $2D$ |
| $\arctan(x)$ | $\tilde{y}_{[d]} = w_{[0]}^{-1} \left( \tilde{x}_{[d]} - \sum_{k=1}^{d-1} w_{[d-k]} \tilde{y}_{[k]} \right)$<br>$\tilde{w}_{[d]} = 2 \sum_{k=1}^{d} x_{[d-k]} \tilde{x}_{[k]}$ | $\sim 2D^2$ | $2D$ |

Table 2.4.: The notation $\tilde{v}_{[j]} := j v_{[j]}$ is used. The results have been adapted from [Griewank and Walther, 2008, Neidinger, 2005]. The zeroth coefficient is computed as $y_{[0]} = \phi(x_{[0]})$. The symbol $\sim$ means *in the highest order*. MOVES is the number of elements that have to be read/written from/to the physical memory.

### 2.3.7. Implicit Functions and Newton-Hensel Lifting

As explained in the previous section, many functions are implicitly defined by equations of the type

$$0 = F(x, y) \in \mathbb{R}^M \ ,$$

where $x \in \mathbb{R}^N$ are the inputs and $y \in \mathbb{R}^M$ the outputs. From the implicit function theorem it is known that if $x(t)$ is smooth and $\partial_y F(x,y) \in \mathcal{L}is(\mathbb{R}^M, \mathbb{R}^M)$ in an open neighborhood of $y$, then it follows that there exists a smooth path $y(t)$ satisfying $0 = F(x(t), y(t))$. It is desired to obtain numerical values for coefficients $y_{[d]}$, $d = 0, \ldots, D-1$. This leads to the task of satisfying

$$0 \overset{D}{=} E_D(F)([x]_D, [y]_D)$$

which are called the *defining equations of order* $D$. These defining equations lead to an algorithmic approach to compute $[y]_D$, the so-called *Newton-Hensel lifting*. The idea is quite simple: Let $[y]_D$ be already known, i.e., it satisfies $0 \overset{D}{=} E_D(F)([x]_D, [y]_D)$. Then one can *lift* the computation to a higher order. Explicitly, one tries to solve $0 \overset{D+E}{=} E_{D+E}(F)([x]_{D+E}, [y]_{D+E})$. The following proposition provides a constructive method to compute the coefficients.

**Proposition 2.3.9** (Newton-Hensel Lifting)**.** *Let $[x]_{D+E}$ and $[y]_D$ be given, $1 \leq E \leq D$ and*

$$0 \overset{D}{=} E_D(F)([x]_D, [y]_D)$$

*be satisfied. Furthermore, assume that $F$ is sufficiently smooth and let $\frac{\partial F}{\partial y}(x_{[0]}, y_{[0]})$ exist and be invertible. Then the coefficients $[\Delta y]_E T^D = [y]_{D+E} - [y]_D$ exist and are defined by*

$$[\Delta y]_E \overset{E}{=} -[F_y]_E^{-1} [\Delta F]_E \ , \tag{2.14}$$

*where $E_{D+E}(F)([x]_{D+E}, [y]_D) \overset{D+E}{=} [\Delta F]_E T^D$ and $[F_y]_E := E_E(\frac{\partial F}{\partial y})([x]_E, [y]_E)$.*

*Proof.* Splitting $[y]_{D+E} = [y]_D + [\Delta y]_E T^D$ and performing a first order Taylor expansion of $F$

about $[y]_D$ yields

$$\begin{aligned}
0 &\overset{D+E}{=} E_{D+E}(F)([x]_{D+E}, [y]_{D+E}) \\
&\overset{D+E}{=} E_{D+E}(F)([x]_{D+E}, [y]_D + [\Delta y]_E T^D) \\
&\overset{D+E}{=} E_{D+E}(F)([x]_{D+E}, [y]_D) + E_E(\frac{\partial F}{\partial y})([x]_E, [y]_E)[\Delta y]_E T^D \\
&\overset{D+E}{=} \underbrace{E_{D+E}(F)([x]_{D+E}, [y]_D)}_{[\Delta F]_E T^D} + E_E(\frac{\partial F}{\partial y})([x]_E, [y]_E)[\Delta y]_E T^D \\
0 &\overset{E}{=} [\Delta F]_E + [F_y]_E [\Delta y]_E
\end{aligned}$$

Since $F$ is sufficiently often differentiable $[F_y]_E$ exists and because of $\frac{\partial F}{\partial y}(x_{[0]}, y_{[0]})$ is invertible it follows also that $[F_y]_E^{-1}$ exists. I.e., since $\overset{E}{=}$ is a "subset" of $\overset{D+E}{=}$ one comes to the conclusion that

$$[\Delta y]_E \overset{E}{=} -[F_y]_E^{-1}[\Delta F]_E \ .$$

□

Setting $E = D$ means that at each step the number of correct coefficients is doubled. In this case it shall be called *Newton's method*. In the case $E = 1$ only the next coefficient is computed. The special case $E = 1$ is called *sequential Hensel lifting*. It is the formula that is typically given as part of the implicit function theorem. The difference is that Newton-Hensel lifting is a purely algebraic task. For a discussion on how to obtain asymptotically fast algorithms and for the nomenclature see for instance [Bernstein, 2001, 2008]. The algorithm is summarized in Algorithm 2.3.1.

**input** : $[x]_D = [x_{[0]}, \ldots, x_{[D-1]}]$, where $x_{[d]} \in \mathbb{R}^N$ for $d = 0, \ldots, D-1$
**output**: $[y]_D = [y_{[0]}, \ldots, y_{[D-1]}]$, where $y_{[d]} \in \mathbb{R}^M$ for $d = 0, \ldots, D-1$

solve $0 = F(x_{[0]}, y_{[0]})$ for $y_{[0]}$

$d = 1$
**while** $d < D$ **do**
  pick some $E$ s.t. $1 \leq E \leq d$
  $[\Delta F]_E T^d \overset{d+E}{=} E_{d+E}(F)([x]_{d+E}, [y]_d)$
  $[F_y]_E \overset{E}{=} E_E(\frac{\partial F}{\partial y})([x]_E, [y]_E)$
  $[\Delta y]_E \overset{E}{=} -[F_y]_E^{-1}[\Delta F]_E$
  $[y]_{d+E} = [y]_d + [\Delta y]_E T^d$
  $d = d + E$
**end**

**Algorithm 2.3.1:** The Newton-Hensel lifting algorithm. At each step one can pick some $1 \leq E \leq d$. The algorithms for explicit choices of $F$ have to provide efficient algorithms for the computation of $[F_y]_E^{-1} \mod T^E$ and $[\Delta F]_E$.

All algorithms in Chapter 3 are derived in this form and may give rise to asymptotically fast algorithms on polynomial matrices. Consider for instance the inversion of $[y]_D = [x]_D^{-1}$, where $[x]_D \in \mathbb{R}[T]/(T^D)$. It is a good example how Newton-Hensel lifting is applied. The defining

equation is

$$0 = xy - 1 \,.$$

Let $y_{[0]} = x_{[0]}^{-1}$ exist and let $0 \overset{D}{=} [x]_D[y]_D - 1$ be satisfied. Then

$$\begin{aligned} & 0 \overset{D+E}{=} [x]_{D+E}([y]_D + [\Delta y]_E T^D) - 1 \\ & \overset{D+E}{=} ([x]_{D+E}[y]_D - 1) + [x]_E[\Delta y]_E T^D \\ \Leftrightarrow \quad & 0 \overset{E}{=} [\Delta F]_E + [x]_E[\Delta y]_E \\ \Leftrightarrow \quad & [\Delta y]_E \overset{E}{=} -[y]_E[\Delta F]_E \end{aligned}$$

One can compute the next $D$ coefficients $[\Delta y]_D$ in $[y]_D$ by two multiplications. This can be done in $\mathcal{O}(D \log D)$ operations if the fast Fourier transform is used. That means the polynomial division is not much more expensive than a polynomial multiplication (c.f. Figure 2.5).

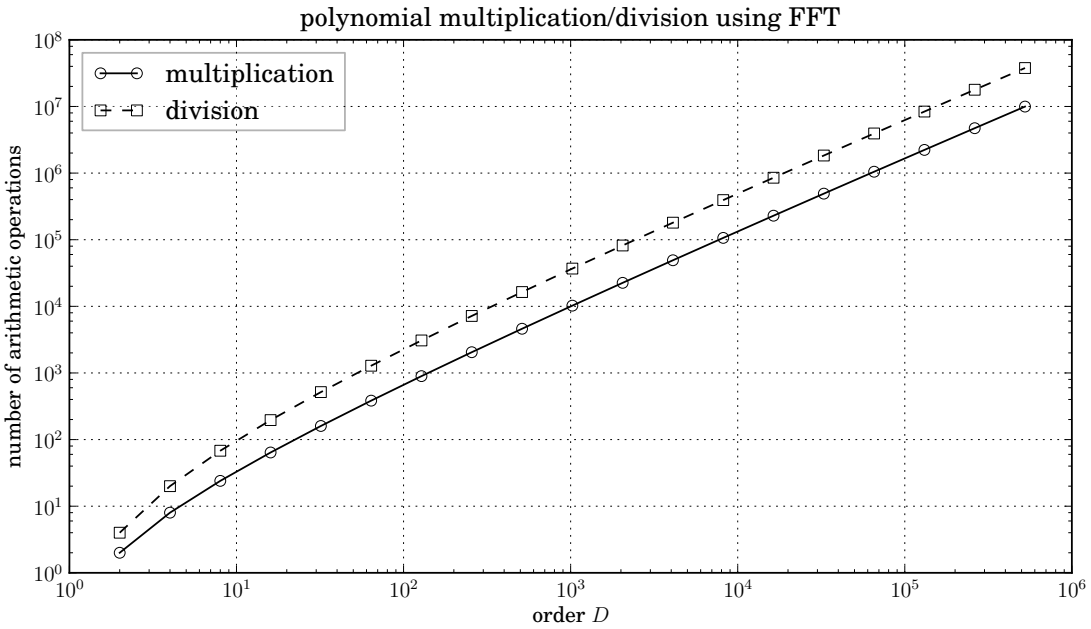



Figure 2.5.: This graph shows the asymptotic complexity of the polynomial multiplication ($2^M M$) to compute $D = 2^M$ coefficients and the asymptotic complexity $\sum_{m=0}^{M} 2^m m$ to compute polynomial division using Newton-Hensel-Lifting in combination with the FFT accelerated multiplication. One can see that the cost of the polynomial division is a small constant multiple of the cost to multiply two polynomials.

```
import numpy; import taylorpoly

def fft_mul(x_data,y_data):
    """ computes [z]_D = [x]_D [y]_D in O( D log(D))"""
    fx = numpy.fft.rfft(x_data, n = x_data.size*2)
    fy = numpy.fft.rfft(y_data, n = y_data.size*2)
    fz = fx * fy
    return numpy.fft.irfft(fz, n = x_data.size*2)[:x_data.size]

def fft_inv(x_data):
    """ computes [y]_D = 1/[x]_D in O( D log(D))"""
    y_data = numpy.zeros_like(x.data)
    y_data[0] = 1./x_data[0]
    d = 1; D = x_data.size
```

```
    while d < D:
        dF = fft_mul(x_data[:2*d], y_data[:2*d])[d:]
        y_data[d:2*d] = - fft_mul(y_data[:d], dF)
        d *= 2

    return y_data

x = taylorpoly.UTPS(numpy.linspace(10,0,8))
y = taylorpoly.UTPS(numpy.linspace(5,1,8))


z1 = taylorpoly.mul(x,y)
z2 = taylorpoly.UTPS(fft_mul(x.data,y.data))
print 'difference fft vs normal mul:\n', z1 - z2

y = taylorpoly.UTPS(fft_inv(x.data))
print ' 1 - x * y = \n', 1 - x*y
```

Listing 2.3: Two FFT accelerated univariate Taylor polynomial algorithms.

```
difference fft vs normal mul:
[  2.84217094e-14   2.84217094e-14  -1.42108547e-14  -1.42108547e-14
   2.84217094e-14   2.84217094e-14  -2.84217094e-14  -1.42108547e-14]
 1 - x * y =
[  0.00000000e+00   1.11022302e-16   2.22044605e-16  -1.11022302e-16
   0.00000000e+00   1.11022302e-16  -8.32667268e-17   4.16333634e-17]
```

Listing 2.4: Output of Listing 2.3.

## 2.4. Forward Mode

In this section it is shown how univariate Taylor polynomial arithmetic can be used to evaluate partial derivatives of the form

$$y_{\mathbf{i}} = \frac{1}{\mathbf{i}!} \frac{\partial^{|\mathbf{i}|}}{\partial z_1^{i_1} \partial z_2^{i_2} \dots \partial z_K^{i_K}} g(x + Sz) \bigg|_{z=0}, \tag{2.15}$$

where $g : \mathbb{R}^N \to \mathbb{R}$ and $z = (z_1, \dots, z_K)^T \in \mathbb{R}^K$. The matrix $S \in \mathbb{R}^{N \times K}$ can be used to compute derivatives restricted to a subspace. The symbol $\mathbf{i} = (i_1, \dots, i_K) \in \mathbb{N}_0^K$ denotes a multi-index. Typically, not just one mixed partial derivative but a set

$$I := \{\mathbf{i}^1, \dots, \mathbf{i}^L\} \tag{2.16}$$

of multi-indices are desired. For instance, the Hessian matrix consists of $L = N(N+1)/2$ distinct elements. More generally, partial derivatives of the form $\{\mathbf{i} : |\mathbf{i}| \leq d\}$ or cross-derivatives of the form $\{\mathbf{i} : i_n \in \{0, 1\}\}$ are desired. Note that $L$ denotes here the number of distinct partial derivatives and not the number of operations. The general approach is illustrated in Figure 2.6. For an overview of this approach see Bischof et al. [1993], where also sparse Hessians and Hessian×vector products are discussed.

### 2.4.1. First-Order Derivatives

It is instructive to consider first-order derivatives before the more general treatment of higher-order derivatives. Often, several directional derivatives have to be computed at once. This situation occurs frequently, for instance when the Jacobian

$$\frac{\partial F}{\partial x}(x) = \left( \frac{\partial F}{\partial x_1}, \dots, \frac{\partial F}{\partial x_N} \right) \in \mathbb{R}^{M \times N}$$

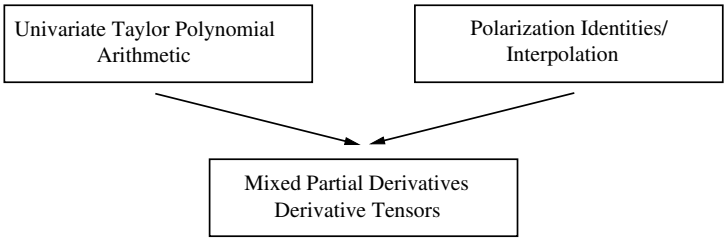



Figure 2.6.: This graph highlights the general idea how to evaluate mixed partial derivatives via a combination of univariate Taylor polynomial arithmetic and interpolation or polarization identities.

of a function $F : \mathbb{R}^N \to \mathbb{R}^M$ with $x \mapsto y = F(x)$ is wanted. The elements $\frac{\partial F}{\partial x_n}$ can be computed by

$$\frac{\partial F}{\partial x_n} = \frac{\partial}{\partial T} F(x + e_n T)\Big|_{T=0} = \frac{\partial F}{\partial x}(x) \cdot e_n \ ,$$

where $e_n, n = 1, \ldots, N$ are the Cartesian basis vectors, i.e.,$e_1 = (1, 0, \ldots, 0)^T$, $e_2 = (0, 1, 0, \ldots)$, etc. To obtain the full Jacobian one could repeat the process $N$ times for each $e_n$. However, in practice such repeated evaluations result in considerable overhead. This is due the fact that the point $x$ at which the directional derivatives are computed stays the same. It is better to propagate several Taylor polynomials at once. Formally, this is written

$$\left(\frac{\partial F}{\partial x}(x)\right) S = \frac{\partial}{\partial T} F(x + ST)\Big|_{T=0} \ , \tag{2.17}$$

where $S \in \mathbb{R}^{N \times P}$ is the so-called *seed matrix*. Choosing the identity matrix $\mathbf{I} \in \mathbb{R}^{N \times N}$ for $S$ results in the complete Jacobian. Now, $\frac{\partial}{\partial T} F(x + e_n T)\Big|_{T=0}$ can be computed by using univariate Taylor polynomial arithmetic. I.e., for $x_{[1]} = e_n$ evaluate

$$[y_{[0]}, y_{[1]}] = E_2(F)([x_{[0]}, x_{[1]}]) \ .$$

In total, this requires to propagate $N$ directions. Later on, it is advantageous to use a more verbose notation for the propagation of several directions. The formula

$$y_{[0]} + \{y_{[1]}\}T = F(x_{[0]} + \{x_{[1]}\}T) + o(T) \tag{2.18}$$

means that a *collection of coefficients*

$$\{x_{[1]}\} := \{x_{[1;1]}, \ldots, x_{[1;P]}\}$$

are propagated. I.e., the notation $x_{[d;p]}$ means the $d$-th coefficient and the $p$-th direction.

**Example 2.4.1.** Consider the function

$$g(x_1, x_2) = x_1 x_2 + x_1 \ . \tag{2.19}$$

It is desired to compute the Jacobian at $x = (3, 7)$. In terms of univariate Taylor polynomial arithmetic one has to evaluate $\frac{\partial f}{\partial T}(x_{[0]} + \mathbf{I}_2 T)\Big|_{T=0}$. According to the discussion in Section 2.3 and Proposition 2.3.3 in particular, it is allowed to decompose the function evaluation into a

concatenation of elementary functions. I.e., using the notation from (2.18), one computes

$$(x_{1;[0]} + \{1,0\}T)(x_{2;[0]} + \{0,1\}T) + (x_{1;[0]} + \{1,0\}T) =$$
$$x_{1;[0]}x_{2;[0]} + \{x_{2;[0]}, x_{1;[0]}\}T + (x_{1;[0]} + \{1,0\}T) + o(T) = x_{1;[0]}x_{2;[0]} + x_{1;[0]} + \{x_{2;[0]} + 1, x_{1;[0]}\}T\ .$$

The expression $x_{1;[0]}x_{2;[0]} + x_{1;[0]}$ is the usual function evaluation and $\{x_{2;[0]} + 1, x_{1;[0]}\}$ the desired Jacobian. The difference to symbolic differentiation is that $x_1$ and $x_2$ are not symbols but numerical values: To show the difference, consider the following program that computes in univariate Taylor polynomial arithmetic.

```
import numpy; numpy.set_printoptions(precision=4, suppress=True)
from taylorpoly.utps import UTPS
x1 = UTPS([3,1,0],P = 2)
x2 = UTPS([7,0,1],P = 2)
print 'x1,x2 = ',x1,x2
v1 = x1*x2; print 'v1 = ', v1
y = v1 + x1; print 'y = ', y
```

Listing 2.5: A simple example that shows that after each step of the computation using univariate Taylor polynomial arithmetic new numerical values are returned.

```
x1,x2 =  [ 3.  1.  0.] [ 7.  0.  1.]
v1 =  [ 21.   7.   3.]
y =  [ 24.   8.   3.]
```

Listing 2.6: Output of Listing 2.5

In other words, after each step in the computation one obtains a numerical, not a symbolical value. In the output $y$ one can see that normal function evaluation is 24 and the gradient is $(8,3)^T$.

### 2.4.2. Computation of the Hessian

First-order derivatives can be extracted easily from the first coefficients of univariate Taylor polynomials (UTPs) as explained above. Things are not quite as simple for higher-order derivatives of multivariate functions. To see what the problem is, consider the computation of the Hessian matrix $H(x) = \nabla_x^2 g(x)$. The elements of the Hessian are $H_{nm} = e_n^T H e_m$ for $n = 1,\ldots,N$ and $m = 1,\ldots,N$. But with univariate Taylor polynomial arithmetic one can only compute diagonal terms

$$\left.\frac{\partial^2 g}{\partial T^2}(x + e_n T)\right|_{T=0} = e_n^T H(x) e_n\ . \tag{2.20}$$

Fortunately, *polarization identities* like

$$s_1^T H s_2 = \frac{1}{2}\left[(s_1+s_2)^T H(s_1+s_2) - s_1^T H s_1 - s_2^T H s_2\right] \tag{2.21}$$

$$\text{or}\quad s_1^T H s_2 = \frac{1}{4}\left[(s_1+s_2)^T H(s_1+s_2) - (s_1-s_2)^T H(s_1-s_2)\right] \tag{2.22}$$

make it possible to recast the task into univariate subtasks. For instance, by application of the polarization identity (2.21), the problem can be reformulated in the following way:

$$\begin{aligned} s_1^T \nabla^2 g(x) s_2 &= \left.\frac{\partial^2 g(x + T_1 s_1 + T_2 s_2)}{\partial T_1 \partial T_2}\right|_{T_1=T_2=0} \\ &= \frac{1}{2}\left[\frac{\partial^2 g(x + T(s_1+s_2))}{\partial T^2} - \frac{\partial^2 g(x+s_1 T)}{\partial T^2} - \frac{\partial^2 g(x+s_2 T)}{\partial T^2}\right]\Bigg|_{T=0}\ . \end{aligned}$$

In other words, it is possible evaluate the mixed partial derivatives via a superposition of UTP coefficients.

**Example 2.4.2.** Let $g : \mathbb{R}^2 \to \mathbb{R}$, $x \mapsto y = g(x) = \sin(x_1 + \cos(x_2)x_1)$. It is necessary to propagate $P = 3$ rays

$$[x]_2 \equiv \underbrace{x_{[0]}}_{\in\mathbb{R}^2} + \underbrace{\begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}}_{=:V\in\mathbb{R}^{2\times P}} T = x_{[0]} + \{x_{[1;1]}, x_{[1;2]}, x_{[1;3]}\}T \, .$$

As one can see, the three directions $x_{[1;p]}$, $p = 1, 2, 3$, are stored in the seed matrix $V$.

```
import numpy; from numpy import sin,cos, array, zeros
from taylorpoly import UTPS
def g(x):
    return sin(x[0] + cos(x[1])*x[0])

def H_fcn(x):
    H11 = -(1+cos(x[1]))**2*sin(x[0]+cos(x[1])*x[0])
    H21 = -sin(x[1]) * cos(x[0] + cos(x[1])*x[0]) \
          +sin(x[1]) *x[0]*(1+ cos(x[1]))*sin(x[0]+cos(x[1])*x[0])
    H22 = -cos(x[1])*x[0]*cos(x[0]+cos(x[1])*x[0])\
          -(sin(x[1])*x[0])**2*sin(x[0]+cos(x[1])*x[0])
    return array([[H11, H21],[H21,H22]])

V = array([[1,0,1],[0,1,1]], dtype=float)
P = V.shape[1]
print 'seed matrix with P = %d directions V = \n'%P, V
x1 = UTPS(zeros(1+2*P), P = P)
x2 = UTPS(zeros(1+2*P), P = P)
x1.data[0] = 3; x1.data[1::2] = V[0,:]
x2.data[0] = 7; x2.data[1::2] = V[1,:]
y = g([x1,x2])
print 'x1=',x1;  print 'x2=',x2; print 'y=',y
H = zeros((2,2),dtype=float)
H[0,0] = 2*y.coeff[0,2]
H[1,0] = H[0,1] = (y.coeff[2,2] - y.coeff[0,2] - y.coeff[1,2])
H[1,1] =  2*y.coeff[1,2]
print 'symbolic Hessian - AD Hessian = \n', H - H_fcn([3,7])
```

Listing 2.7: Interpolation of the Hessian of the function defined in line 3 by application of the polarization identity (2.21).

```
seed matrix with P = 3 directions V =
[[ 1.  0.  1.]
 [ 0.  1.  1.]]
x1= [ 3.  1.  0.  0.  0.  1.  0.]
x2= [ 7.  0.  0.  1.  0.  1.  0.]
y= [-0.85288091  0.91572201  1.31180466 -1.02904895  1.06616101 -0.11332694
 -0.91334978]
symbolic Hessian - AD Hessian =
[[  0.00000000e+00   0.00000000e+00]
 [  0.00000000e+00  -4.44089210e-16]]
```

Listing 2.8: Output of Listing 2.7. The solution of the "interpolated" Hessian is compared to the symbolically derived Hessian and one finds that it is correct close to the machine precision.

### 2.4.3. Higher-Order Partial Derivatives via Polarization Formulas

Polarization identities for multilinear maps are called polarization formulas. They can be used to evaluate higher-order derivatives by application of univariate Taylor polynomial arithmetic, similarly to the interpolation of the Hessian. They can be found in proofs for theorems. E.g.,

Nguyen [2009] uses the polarization identity

$$\partial^d g(x)\{v_1,\ldots,v_d\} = \frac{1}{2^K d!} \sum_{e_1,\ldots,e_d\in\{\pm 1\}} e_1 \ldots e_d g^{(d)}(x; e_1 v_1 + \cdots + e_d v_d) \tag{2.23}$$

to derive results for the radius of analyticity of Taylor series in real Banach spaces. $g^{(d)}(x;w)$ is a homogeneous polynomial in $w$ of degree $d$. Or, as remarked by Thomas, to prove that for multivariate normally distributed random variables $x_1,\ldots,x_N$ the expectation value can be written as a sum of products $\mathbb{E}[x_1\cdots x_N] = \sum\prod \mathbb{E}[x_i x_j]$ when $N$ is even. In [Lucien, 1971] one can find an introductory discussion of polarization formulas. Let $\hat{u} : \mathbb{R}^N \times \cdots \times \mathbb{R}^N \to \mathbb{R}^M$ be a symmetric $K-$linear mapping and let $u(x) := \hat{u}(x,\ldots,x)$, then a polarization identity allows one to recover $\hat{u}$ when $u$ is given by application of the formula

$$K!\hat{u}(e_1,\ldots,e_K) = \sum_{\mathbf{i}\subseteq\{1,\ldots,K\}} (-1)^{K-c(\mathbf{i})} u(\sum_{\mathbf{j}\in\mathbf{i}} e_{\mathbf{j}}) \,, \tag{2.24}$$

where $c(\mathbf{i})$ is the number of elements of $\mathbf{i}$.

Such polarization identities are not only interesting as a theoretical tool but also useful for the evaluation of higher-order derivatives and the evaluation of multivariate Taylor polynomials. Much work has been done by Griewank et al. [2000, 2009] and Neidinger [2005], Altman [2010], who call their methods *interpolation techniques*. This stems from the fact that in the process multivariate polynomials are interpolated by sufficiently many support points. Griewank therefore calls the approach *exact interpolation*.

As already stated in the introduction of this section, one would like to compute multivariate Taylor polynomial coefficients

$$y_{\mathbf{i}} = \frac{1}{\mathbf{i}!} \frac{\partial^{|\mathbf{i}|}}{\partial z_1^{i_1} \partial z_2^{i_2} \ldots \partial z_K^{i_K}} g(x+Sz)\bigg|_{z=0} \tag{2.25}$$

of the $D-1$ times continuously differentiable function $g : \mathbb{R}^N \to \mathbb{R}$, where $z = (z_1,\ldots,z_K)^T \in \mathbb{R}^K$. As one can see, one can simply scale $y_{\mathbf{i}}$ by $\mathbf{i}!$ to obtain partial derivatives. One defines

$$\begin{aligned} f : \mathbb{R}^K &\to \mathbb{R} \\ z &\mapsto y = f(z) := g(x+Sz) \,, \end{aligned}$$

since it simplifies the notation. The function $f(tz)$ can be expanded in $t$ as follows:

$$f(tz) = \sum_{d=0}^{D-1} \underbrace{\frac{1}{d!}\frac{\partial^d}{\partial t^d} f(tz)\bigg|_{t=0}}_{=:f^{(d)}(0;z)} t^d + o(|tz|^{D-1}) \,.$$

$f^{(d)}(z)$ is a homogeneous polynomial in $z$ of order $d$. On the other hand, application of the multivariate Taylor series expansion in $z$ yields

$$\begin{aligned} f(tz) &= \sum_{|\mathbf{i}|<D} y_{\mathbf{i}} (zt)^{\mathbf{i}} + o(|zt|^{D-1}) \\ &= \sum_{d=0}^{D-1} \sum_{|\mathbf{i}|=d} y_{\mathbf{i}} z^{\mathbf{i}} t^d + o(|zt|^{D-1}) \end{aligned}$$

where $y_{\mathbf{i}} = \frac{1}{\mathbf{i}!}\frac{\partial^{|\mathbf{i}|}}{\partial z^{\mathbf{i}}} f(z)\Big|_{z=0}$. Hence, one comes to the conclusion that the identity

$$\sum_{|\mathbf{i}|=d} y_{\mathbf{i}} z^{\mathbf{i}} = \frac{1}{d!}\frac{\partial^d}{\partial t^d} f(zt)\Big|_{t=0} \tag{2.26}$$

holds. I.e., on the lhs are the wanted coefficients $y_{\mathbf{i}}$ and on the rhs coefficients of a univariate Taylor polynomial. In total, there are more unknowns than equations. To obtain a linear system of the form $Ax = b$ with full rank, it is necessary to choose sufficiently many suitable $z$'s and evaluate at least

$$M \geq L = |\{\mathbf{i} = (i_1, \ldots, i_K) \in \mathbb{N}_0^K : |\mathbf{i}| = d\}| = \binom{K+d-1}{d}$$

UTP coefficients of degree $d$. Then, it is possible to "*interpolate*" $y_{\mathbf{i}}$ by solving the linear system

$$Gy = b \,,$$

where $G \in \mathbb{R}^{M\times L}$, $y \in \mathbb{R}^L$ and $b \in \mathbb{R}^M$ with elements

$$G_{ij} = (z^{(i)})^{\mathbf{j}} \,, \quad y_j = y_{\gamma(\mathbf{j})} \,, \quad b_i = f^{(d)}(z^{(i)}) \,.$$

In the above equation it is assumed that there is a one-to-one mapping

$$\begin{aligned} \gamma : \mathbb{N}_0^K &\to \mathbb{N} \\ \mathbf{i} &\mapsto i = \gamma(\mathbf{i}) \,, \end{aligned}$$

i.e., $j = \gamma(\mathbf{j})$ and $i = \gamma(\mathbf{i})$ and there are $M$ suitable choices $z^{(1)}, z^{(2)}, \ldots, z^{(M)}$.

**Example 2.4.3.** It is the task to compute the Hessian of $g(x) = x_1x_2x_3 + x_1^2x_2 + x_3x_2$ by the above evaluation/interpolation approach at the point $x = (1,2,3)^T \in \mathbb{R}^3$, i.e., $\nabla_z^2 g(x+z)\big|_{z=0}$ and $K = 3$. For such a simple function it is easy to find the analytical Hessian which is given as

$$H(x) = \nabla_x^2 g(x) = \begin{pmatrix} 2x_2 & x_3 + 2x_1 & x_2 \\ & 0 & x_1 + 1 \\ & & 0 \end{pmatrix} .$$

Due to the symmetry, only the upper triangular part is shown. The Hessian has $L = \binom{4}{2} =$ 6 distinct elements and therefore $M = L = 6$ suitable choices for $z^{(m)}$, $m = 1, \ldots, M$ are required. When $z^{(m)}$ is chosen randomly one already obtains a matrix $G$ with acceptable condition number. See Listing 2.9 and its output shown in Listing 2.10. One can see that the coefficients $y_{\mathbf{i}}$ are correct. Note for that the coefficients are scaled by $\frac{1}{|\mathbf{i}|}$ which explains the discrepancy between $H_{11} = 4$ and $y_{(2,0,0)}$.

```
import numpy; numpy.set_printoptions(precision=4, suppress=True)
import algopy.exact_interpolation as exint

def g(x):
    return x[0]*x[1]*x[2] + x[0]**2*x[1] + x[2]*x[1]

def hess(x):
    N = numpy.size(x)
    retval = numpy.zeros((N,N))
    retval[0,:] = [2*x[1], x[2] + 2*x[0], x[1]]
    retval[1,:] = [0  , 0  , x[0] + 1]
    retval[2,:] = [0, 0, 0 ]
```

```
    return retval

# setup interpolation
N,D = 3, 5
deg = 2
I = exint.generate_multi_indices(N,deg)
M = I.shape[0]
V = numpy.random.rand(N,M)
G = numpy.zeros((M,M))
for m in range(M):
    G[m,:] = exint.multi_index_pow(V[:,m], I)

# UTP computation
from algopy import UTPM
x = UTPM(numpy.zeros((D,M,N)))
x.data[0,:,:] = [1,2,3]
x.data[1,:,:] = V.T
y = g(x)

# compute interpolation
y2 = numpy.linalg.solve(G,y.data[deg])

# output
print 'singular values of G = ', numpy.linalg.svd(G)[1]
print 'V=\n',V
print 'I=\n',I
print 'y_i, |i|=2 is =\n', y2
print 'symbolic Hessian = ', hess([1,2,3])
```

Listing 2.9: This listing shows a complete example how all partial derivatives of the Hessian can be computed by the interpolation approach.

```
$ python general_interpolation.py
singular values of G =  [ 1.652   1.2055  0.4388  0.138   0.0443  0.0227]
V=
[[ 0.3522  0.8611  0.1688]
 [ 0.6799  0.0989  0.4527]
 [ 0.9404  0.1668  0.7358]
 [ 0.2301  0.2834  0.6798]
 [ 0.2518  0.7327  0.3393]
 [ 0.5244  0.9265  0.1188]]
I=
[[2 0 0]
 [1 1 0]
 [1 0 1]
 [0 2 0]
 [0 1 1]
 [0 0 2]]
y_i, |i|=2 is =
[ 2.  5.  2.  0.  2.  0.]
symbolic Hessian =  [[ 4.  5.  2.]
 [ 0.  0.  2.]
 [ 0.  0.  0.]]
```

Listing 2.10: Output of Listing 2.9.

**Example 2.4.4** (reconstruction of a polynomial)**.** It is the goal to reconstruct a multivariate polynomial

$$g(x) = 3 + 7x_1x_2 + 0.5x_1 + 17x_1x_2^2 + 13x_1^2x_2$$

by propagation of univariate Taylor polynomials.

```
import algopy; from algopy import UTPM
import algopy.exact_interpolation as exint
import numpy; numpy.set_printoptions(precision=4, suppress=True)

def g(x):
    return 3 + 7*x[0]*x[1] + 0.5*x[0] + 17*x[0]*x[1]**2 + 13*x[0]**2*x[1]

# setup interpolation
N,D = 2, 7
```

```
def generate_matrices(N, deg):
    I = exint.generate_multi_indices(N,deg)
    L = I.shape[0]
    V = numpy.random.rand(N,L)
    G = numpy.zeros((L,L))
    for l in range(L):
        G[l,:] = exint.multi_index_pow(V[:,l], I)

    return G, V, I, L

for deg in [0,1,2,3]:
    print 'degree =', deg

    # setup rays and interpolation matrix G
    G, V, I, L = generate_matrices(N, deg)

    # UTP computation
    x = UTPM(numpy.zeros((D,L,N)))
    x.data[1,:,:] = V.T
    y = g(x)

    # compute interpolation
    y2 = numpy.linalg.solve(G,y.data[deg])

    for l in range(L):
        print 'y_(%s) = %+f' %(str(I[l]),y2[l])
```

Listing 2.11: This listing shows a complete example how all coefficients of a multivariate polynomial can be reconstructed from UTPs.

```
python reconstruction_of_a_polynomial.py
degree = 0
y_([0 0]) = +3.000000
degree = 1
y_([1 0]) = +0.500000
y_([0 1]) = +0.000000
degree = 2
y_([2 0]) = +0.000000
y_([1 1]) = +7.000000
y_([0 2]) = +0.000000
degree = 3
y_([3 0]) = -0.000000
y_([2 1]) = +13.000000
y_([1 2]) = +17.000000
y_([0 3]) = +0.000000
```

Listing 2.12: Output of Listing 2.11. As one can see from the output it is possible to compute multivariate coefficients by interpolation of univariate coefficients. Note that in the above code explicit matrices $G$ have to be constructed for all degrees $d \in \{0, 1, 2, 3\}$.

### 2.4.4. Griewank-Utke-Walther Interpolation

Higher-order partial derivatives can be computed using generalization of the polarization identities of the previous section to symmetric multilinear forms. This is a consequence of the fact that there exists an isomorphism between $d-$symmetric multilinear maps and homogoneneous polynomials of degree $d$. Griewank et al. [2000] have shown that it is possible to interpolate all coefficients

$$y_{\mathbf{i}} = \frac{1}{\mathbf{i}!} \frac{\partial^{|\mathbf{i}|}}{\partial z_1^{i_1} \partial z_2^{i_2} \dots \partial z_K^{i_K}} f(z) \Big|_{z=0}$$

with $|\mathbf{i}| \leq d$ ($\mathbf{i} \in \mathbb{N}_0^K$, $z \in \mathbb{R}^K$) by propagating much fewer directions compared to the approach in the previous section. In formulas, $\binom{K+d-1}{d}$ compared $\binom{K+d}{d}$. Furthermore, the task of inverting the matrix $G$ is not necessary. More explicitly, the method yields all coefficients that are necessary to describe a multivariate Taylor polynomial of order $d$ by propagation of $N_{|\mathbf{i}|=d}$

univariate Taylor polynomials of order $d$. It will be referred to as the Griewank-Utke-Walther (GUW) interpolation method.

**Lemma 2.4.1.** *1. The set* $I = \{\mathbf{i} : |\mathbf{i}| = d, \mathbf{i} \in \mathbb{N}_0^K\}$ *has the size*

$$N_{|\mathbf{i}|=d} := |I| = \binom{K+d-1}{d} .$$

*2. The set* $I = \{\mathbf{i} : |\mathbf{i}| \leq d, \mathbf{i} \in \mathbb{N}_0^K\}$ *has the size*

$$N_{|\mathbf{i}|\leq d} := |I| = \sum_{k=0}^{d} \binom{K+k-1}{k} = \binom{K+d}{d} . \tag{2.27}$$

*Proof.* The first formula follows from combinatorial arguments. The second by induction

- base case: $d = 0$, then $\sum_{k=0}^{d} \binom{K+k-1}{k} = \binom{K+d}{d} = 1$.
- induction assumption: $\sum_{k=0}^{d-1} \binom{K+k-1}{k} = \binom{K+d-1}{d-1}$ has already been verified.
- induction step: advance from $d-1$ to $d$ and use the induction assumption:

$$\begin{aligned}
\sum_{k=0}^{d} \binom{K+k-1}{k} =& \sum_{k=0}^{d-1} \binom{K+k-1}{k} + \binom{K+d-1}{d} \\
=& \binom{K+d-1}{d-1} + \binom{K+d-1}{d} \\
=& \frac{(K+d)!}{K!d!} \frac{K!d!}{(K+d)!} \left( \frac{(K+d-1)!}{K!(d-1)!} + \frac{(K+d-1)!}{(K-1)!d!} \right) \\
=& \binom{K+d}{d} .
\end{aligned}$$

□

**Proposition 2.4.2.** *Let the function*

$$\begin{aligned}
f : \mathbb{R}^K \to& \mathbb{R} \\
z \mapsto& y = f(z) ,
\end{aligned}$$

*be* $d$ *times continuously differentiable in an open neighborhood* $U$ *of* $0 \in \mathbb{R}^K$*. Then the equality*

$$y_{\mathbf{i}} = \frac{1}{\mathbf{i}!} \frac{\partial^{|\mathbf{i}|} f}{\partial z^{\mathbf{i}}}(z)\bigg|_{z=0} = \frac{1}{\mathbf{i}!} \sum_{0 \leq \mathbf{k} \leq \mathbf{i}} \binom{\mathbf{i}}{\mathbf{k}} (-1)^{|\mathbf{i}-\mathbf{k}|} f^{(|\mathbf{i}|)}(0; \mathbf{k}) , \tag{2.28}$$

*where* $\mathbf{k} \in \mathbb{N}_0^K$*, holds for all* $|\mathbf{i}| \leq d$*,* $\mathbf{i} \in \mathbb{N}_0^K$*.*

*Proof.*

$$\begin{aligned}
\left.\frac{\partial^{|\mathbf{i}|} f(z)}{\partial z^{\mathbf{i}}}\right|_{z=0} &= \left.\frac{\partial^{|\mathbf{i}|} f(tz)}{\partial z^{\mathbf{i}}}\right|_{t=1,z=0} \\
&= \left.\frac{\partial^{|\mathbf{i}|} \sum_{d=0}^{|\mathbf{i}|} \frac{1}{d!} \frac{\partial^d}{\partial t^d} f(tz)\Big|_{t=0} t^d}{\partial z^{\mathbf{i}}}\right|_{t=1,z=0} \\
&= \left.\frac{\partial^{|\mathbf{i}|} \sum_{d=0}^{|\mathbf{i}|} \frac{1}{d!} \partial^d f(0)\{z,\dots,z\}}{\partial z^{\mathbf{i}}}\right|_{z=0} \\
&= \underbrace{\frac{\partial^{|\mathbf{i}|}}{\partial z^{\mathbf{i}}} f^{(|\mathbf{i}|)}(0;z)}_{=\text{const.}} \\
&= \frac{1}{(1-0)^{|\mathbf{i}|}} \int_{[0,1]^{|\mathbf{i}|}} \frac{\partial^{|\mathbf{i}|}}{\partial z^{\mathbf{i}}} f^{(|\mathbf{i}|)}(0;z)\mathrm{d}z \\
&= \sum_{0\le\mathbf{k}\le\mathbf{i}} \binom{\mathbf{i}}{\mathbf{k}} (-1)^{|\mathbf{i}-\mathbf{k}|} f^{(|\mathbf{i}|)}(0;\mathbf{k}) \ .
\end{aligned}$$

It has been made use of the fact that $\partial_x(\frac{1}{2}\partial_x f(x)h)h \equiv \frac{1}{2}\partial_x^2 f(x)\{h,h\} =: f^{(2)}(x;h)$ (c.f. Section 2.2). The sum $\sum_{d=0}^{|\mathbf{i}|}$ can be dropped since $\left.\frac{\partial^{|\mathbf{i}|}}{\partial z^{\mathbf{i}}} f^{(d)}(0;z)\right|_{z=0}$ is zero for all $d = 0,\dots,|\mathbf{i}|-1$. One can check that for any polynomial $P$ of degree $K$ or less

$$\frac{\partial^K P(z)}{\partial z_1 \dots \partial z_K} = \sum_{i_1=0}^{1} \cdots \sum_{i_K=0}^{1} P(i_1 e_1 + \cdots + e_K i_K)(-1)^{K-(i_1+\cdots+i_K)}$$

by integration of the constant function on the left hand side over the unit cube in $K$ dimensions, and more generally

$$\frac{\partial^{|\mathbf{i}|} P(z)}{\partial z_1^{i_1} \dots \partial z_K^{i_K}} = \sum_{j_1=0}^{i_1} \cdots \sum_{j_K=0}^{i_K} \binom{i_1}{j_1} \dots \binom{i_K}{j_K} (-1)^{|\mathbf{i}-\mathbf{k}|} P(\mathbf{k}) \ .$$

□

This Proposition can already be used to compute all partial derivatives described by a set $I$ as defined in (2.16). To compute all partial derivatives

$$I = \{\mathbf{i} \in \mathbb{N}_0^K : |\mathbf{i}| \le d\} \ .$$

it is not necessary to repeat the process for $d = 0, 1, \dots$. Looking at (2.28) one can see that many of the $\mathbf{k}$ would be used repeatedly. I.e., the same $\mathbf{k}$ would show up for different $\mathbf{i} \in I$. Additionally to this observation one can use the fact one can interpolate polynomials from sufficiently many evaluations as the following results show.

**Lemma 2.4.3** (Evaluation/Interpolation for polynomials)**.** *Let $p : \mathbb{R}^K \to \mathbb{R}$, $z \mapsto p(z)$ be a polynomial of degree $d$ and $K \le d$. Then*

$$p(z) = \sum_{|\mathbf{j}|=d} \binom{z}{\mathbf{j}} p(\mathbf{j}) \ , \tag{2.29}$$

*where $\mathbf{j} \in \mathbb{N}_0^K$ holds for all $z \in \mathbb{R}^K$ s.t. $\sum_{k=1}^K z_k = d$.*

*Proof.* This result is implicitly used in the proof of [Griewank et al., 2000] and [Griewank and Walther, 2008, Chapter 13.3] but it is not elaborated upon. R. Neidinger suggests (in personal communication) to prove this Lemma as follows: Since $|z| \equiv \sum_{k=1}^{K} z_k = d$ per assumption of the Lemma, and using $z \equiv (x, d - |x|)$ resp. $\mathbf{j} \equiv (\mathbf{i}, d - |\mathbf{i}|)$, one may rewrite the claim of the Lemma as

$$p(x, d - |x|) = \sum_{|\mathbf{i}| \leq d} \binom{x}{\mathbf{i}} \binom{d - |x|}{d - |\mathbf{i}|} p(\mathbf{i}, d - |\mathbf{i}|)$$

where $|x| = \sum_{k=1}^{K-1} x_k \leq d$. The left hand side (lhs) as well as the right hand side (rhs) is a polynomial of degree $\leq d$ in $K - 1$ variables. It is known from classical theory of interpolation (e.g. Gasca and Sauer [2000] and references therein), that such a polynomial is uniquely defined by the function values on the nodes $\{\mathbf{i} : |\mathbf{i}| \leq d\}$. I.e., if one can show that they agree on this set of nodes, one has shown that they are in fact the same polynomial. But this follows easily by looking again at the claim of the Lemma, because $\binom{\mathbf{k}}{\mathbf{j}} = \delta_{\mathbf{kj}}$ for $|\mathbf{k}| = |\mathbf{j}|$.

□

**Theorem 2.4.4** (Exact Interpolation)**.** *Let the function*

$$\begin{aligned} f : \mathbb{R}^K &\to \mathbb{R} \\ z &\mapsto f(z) \end{aligned}$$

*be $d$ times continuously differentiable in an open neighborhood $U$ of $0$. Then for any $\mathbf{i} \in \mathbb{N}_0^K$ with $1 \leq |\mathbf{i}| \leq d$ the coefficient $y_{\mathbf{i}}$ can be computed by*

$$y_{\mathbf{i}} = \frac{1}{\mathbf{i}!} \frac{\partial^{|\mathbf{i}|} f}{\partial z^{\mathbf{i}}}(z)\bigg|_{z=0} = \sum_{|\mathbf{j}|=d} \gamma_{\mathbf{i},\mathbf{j}} f^{(|\mathbf{i}|)}(0; \mathbf{j}) \,, \tag{2.30}$$

$$\gamma_{\mathbf{i},\mathbf{j}} = \frac{1}{\mathbf{i}!} \sum_{0 \leq \mathbf{k} \leq \mathbf{i}} (-1)^{|\mathbf{i}-\mathbf{k}|} \binom{\mathbf{i}}{\mathbf{k}} \binom{\frac{d\mathbf{k}}{|\mathbf{k}|}}{\mathbf{j}} \left(\frac{|\mathbf{k}|}{d}\right)^{|\mathbf{i}|} \,, \tag{2.31}$$

*where $\mathbf{i}, \mathbf{j}, \mathbf{k} \in \mathbb{N}_0^K$ are multi-indices.*

*Proof.* One can transform (2.28) as follows:

$$\begin{aligned} y_{\mathbf{i}} =& \frac{1}{\mathbf{i}!} \sum_{0 \leq \mathbf{k} \leq \mathbf{i}} \binom{\mathbf{i}}{\mathbf{k}} (-1)^{|\mathbf{i}-\mathbf{k}|} f^{(|\mathbf{i}|)}(0; \mathbf{k}) \\ =& \frac{1}{\mathbf{i}!} \sum_{0 \leq \mathbf{k} \leq \mathbf{i}} \binom{\mathbf{i}}{\mathbf{k}} (-1)^{|\mathbf{i}-\mathbf{k}|} \left(\frac{|\mathbf{k}|}{d}\right)^{|\mathbf{i}|} f^{(|\mathbf{i}|)}(0; \mathbf{k}\frac{d}{|\mathbf{k}|}) \\ =& \frac{1}{\mathbf{i}!} \sum_{0 \leq \mathbf{k} \leq \mathbf{i}} \binom{\mathbf{i}}{\mathbf{k}} (-1)^{|\mathbf{i}-\mathbf{k}|} \left(\frac{|\mathbf{k}|}{d}\right)^{|\mathbf{i}|} \sum_{|\mathbf{j}|=d} \binom{\mathbf{k}\frac{d}{|\mathbf{k}|}}{\mathbf{j}} f^{(|\mathbf{i}|)}(0; \mathbf{j}) \\ =& \sum_{|\mathbf{j}|=d} \left( \frac{1}{\mathbf{i}!} \sum_{0 \leq \mathbf{k} \leq \mathbf{i}} \binom{\mathbf{i}}{\mathbf{k}} (-1)^{|\mathbf{i}-\mathbf{k}|} \left(\frac{|\mathbf{k}|}{d}\right)^{|\mathbf{i}|} \binom{\mathbf{k}\frac{d}{|\mathbf{k}|}}{\mathbf{j}} \right) f^{(|\mathbf{i}|)}(0; \mathbf{j}) \,. \end{aligned}$$

From the second to the third line Lemma 2.4.3 has been applied. □

In other words, to compute partial derivatives corresponding to the multi-indices $\mathbf{i} \in \{\mathbf{i} : |\mathbf{i}| \leq d\}$ one has to compute

$$E_{d+1}(f)([x_0, \{S\mathbf{j} : |\mathbf{j}| = d, \mathbf{j} \in \mathbb{N}_0^K\}, 0, \ldots, 0])$$

i.e., $\binom{K+d-1}{d}$ directions.

**Example 2.4.5.** Consider the polynomial

$$p(x) = 7x_1x_2 + \frac{1}{2}x_1 + 17x_1x_2^2 + 13x_1^2x_2$$

and treat it as if the coefficients were not known. I.e., here $K = 2$ and $d = 3$. The following listing shows how the coefficients can be interpolated from numerical values obtained by UTP arithmetic.

```
import algopy; from algopy import UTPM
import algopy.exact_interpolation as exint
import numpy; numpy.set_printoptions(precision=4, suppress=True)

def f(x):
    return 7*x[0]*x[1] + 0.5*x[0] + 17*x[0]*x[1]**2 + 13*x[0]**2*x[1]

# setup interpolation
K,D = 2, 4

J = exint.generate_multi_indices(K,D-1); L = J.shape[0]
print 'J=',J

# UTP computation
x = UTPM(numpy.zeros((D,) + J.shape))
x.data[1,:,:] = J
y = f(x)

for deg in [1,2,3]:
    print 'degree =', deg
    # GUW interpolation
    I = exint.generate_multi_indices(K,deg)
    for i in I:
        y_i = sum([ exint.gamma(i,J[l])*y.data[deg,l] for l in range(L) ])
        print 'y_(%s) = %+f' %(str(i), y_i)
```

The program indeed returns the correct coefficients.

```
J= [[3 0]
 [2 1]
 [1 2]
 [0 3]]
degree = 1
y_([1 0]) = +0.500000
y_([0 1]) = +0.000000
degree = 2
y_([2 0]) = +0.000000
y_([1 1]) = +7.000000
y_([0 2]) = +0.000000
degree = 3
y_([3 0]) = +0.000000
y_([2 1]) = +13.000000
y_([1 2]) = +17.000000
y_([0 3]) = +0.000000
```

### Complexity of Derivative Tensor Computation

One can use Theorem 2.4.4 to compute higher-order derivative tensors of functions with many inputs, but one should note that this quickly gets very time consuming. On the other hand, compared to direct application of multivariate Taylor polynomial arithmetic this approach scales favorably. This is discussed below.

The computational cost to apply GUW interpolation depends on the number of elements in $|\mathbf{j}| = d$, $\gamma_{\mathbf{i},\mathbf{j}}$ and the number of operations needed in Taylor arithmetic. The number of distinct elements in derivative tensor $\partial^d f$ is given by Lemma 2.4.1 and denoted $q(d,K) = \frac{K+d-1}{d}$ and the approximate cost to compute $E_D(f)$ is roughly $s(d) := \text{ops}(f)d^2$. Hence, the total

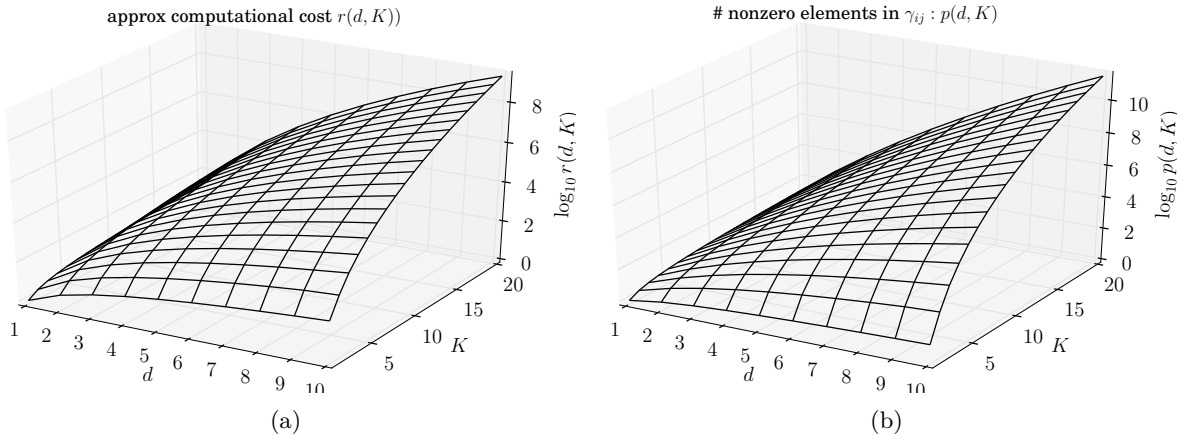
computational cost to propagate all rays is

$$r(d, K) := q(d, K)d^2\text{ops}(f) \ .$$

Once all rays are computed, it is necessary to interpolate the derivative tensor. The cost of this depends on the number of nonzero elements $\gamma_{\mathbf{i},\mathbf{j}}$. As shown in [Griewank et al., 2000] it is possible to find an upper bound

$$p(d, K) = \sum_{k=1}^{d} \binom{K}{k} \binom{d}{k} \binom{k+d-1}{d}$$

for the number of $\gamma_{\mathbf{i},\mathbf{j}}$ that are nonzero. Both $r(d, K)$ and $p(d, K)$ are of moderate size when either $N$ or $d$ is small. E.g., $p(4, 3) = 162$, $r(4, 3) = 240$, $p(8, 2) = 268$, $r(8, 2) = 576$. But in general, the complexity grows rapidly, e.g., $p(6, 10) = 750915$ and $r(6, 10) = 180180$. This behavior is shown in Figure 2.7.




(a) (b)

Figure 2.7.: In (a) the approximate computational cost to compute derivative tensors w.r.t. $K$ independent variables to degree $d$ is shown. One can see that higher-order derivatives of functions with many inputs $K \gg 1$ is very expensive.

Now to the question how the GUW interpolation compares to multivariate Taylor polynomial arithmetic. A multivariate polynomial multiplication

$$\sum_{0 \le \mathbf{i} \le d} z_{\mathbf{i}} T^{\mathbf{i}} = \left( \sum_{0 \le \mathbf{i} \le d} x_{\mathbf{i}} T^{\mathbf{i}} \right) \left( \sum_{0 \le \mathbf{i} \le d} y_{\mathbf{i}} T^{\mathbf{i}} \right) + \text{higher-order terms}$$

requires $A(d, K)$ multiplications, where

$$A(d, K) := \binom{2K+d}{d} \approx \frac{(2K)^d}{d!} \quad \text{if} \quad d \ll K \ .$$

Propagation of $\binom{K+d-1}{d}$ directions, each requiring $\binom{d+2}{d}$ multiplications requires in total

$$B(d, K) := \binom{K+d-1}{d} \binom{d+2}{d}$$

multiplications. One can show that the ratio

$$q(d,K) := \frac{B(d,K)}{A(d,K)} = \frac{(d+2)(d+1)}{2} \frac{(K+d-1)\dots K}{(2K+d)(2K+d-1)\dots(2K+1)} \tag{2.32}$$

is never worse than $\frac{3}{2}$. In Figure 2.8 one can see plots for different values of $K$. I.e., univariate Taylor polynomial arithmetic combined with polarization/interpolation scales favorably with $K$ and degree $d$ when compared to multivariate Taylor polynomial arithmetic, at least in theory. Griewank et al. [2000] also argues that the univariate Taylor polynomial approach should also more efficient in practice due to simpler memory access patterns.

*Remark.* The theory suggests that the propagation of the univariate Taylor polynomials should be computationally more expensive than the interpolation, granted the function is sufficiently complex. One should note that the direct implementation of the formula from Theorem 2.4.4 in an interpreted language such as Matlab can be rather slow and therefore be a bottleneck [Altman, 2010].

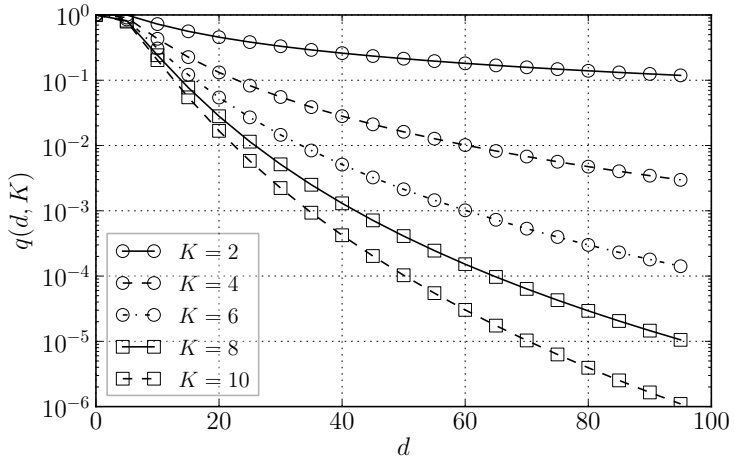



Figure 2.8.: The ratio $q(d,K)$ from (2.32). Please note that the operations to perform the GUW interpolation are not included in $q(d,K)$, because its cost is very small in comparison when $\mathrm{ops}(f) \gg 1$.

### 2.4.5. Second-Order Cross-Derivatives

In many cases one is not interested in all multivariate Taylor polynomial coefficients to a certain degree. E.g., one would only like to know certain elements of the Hessian matrix. An important case is when an off-diagonal block has to be computed. This is a special case of a cross-derivative, of which the general case is discussed in [Griewank et al., 2009].

**Definition 2.4.1** (cross-derivative)**.** Let $f : \mathbb{R}^K \to \mathbb{R}$ be $d$-times continuously differentiable. A partial derivative

$$\frac{\partial^{|\mathbf{i}|} f}{\partial x^{\mathbf{i}}}(x) ,$$

where $\mathbf{i} \in \mathbb{N}_0^K$ is a multi-index is called a cross-derivative if and only if

$$i_K \leq 1 \quad \text{for all} \quad k = 1, \dots, K .$$

Consider the following example: Let $f : \mathbb{R}^{N+M} \to \mathbb{R}$, $x \mapsto y = f(x)$ with $N = 3, M = 2$ and

thus $K = N + M = 5$. The partial derivatives are defined by the set of multi-indices

$$\begin{pmatrix} 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix}$$

are wanted. E.g., the first row means to differentiate the function once w.r.t. the first and fourth variable. GUW interpolation would require $\binom{N+M+1}{2} = 15$ rays that correspond to the computation of $\binom{N+M+1}{2}$ distinct elements of the Hessian matrix. Using the other polarization identity (2.22) only $2MN = 12$ directions have to be propagated and using the polarization identity (2.21) would require the propagation of $N + M + NM = 11$ directions.

To make the example explicit how (2.22) can be used, consider Listing 2.13.

```
import algopy
def f(x):
    return x[0]*x[4] + x[1]*x[3]**2 + algopy.cos(x[2])/x[4]

I = numpy.eye(5)
H = numpy.ones((5,5))*numpy.nan
x = algopy.UTPM(numpy.zeros((3,1,5)))
x.data[0] = 3.
for i in range(3):
    for j in range(3,5):
        s1 = I[:,i]; s2 = I[:,j]
        x.data[1,0,:] = s1+s2; y1 = f(x)
        x.data[1,0,:] = s1-s2; y2 = f(x)
        H[i,j] = (y1-y2).data[2,0]/2.
print 'offdiagonal part of the Hessian = \n',H
```

Listing 2.13: The program computes the offdiagonal part of the Hessian using (2.22). The elements of the Hessians that are not computed are set to NaN.

```
offdiagonal part of the Hessian =
[[      NaN       NaN       NaN  0.        1.      ]
 [      NaN       NaN       NaN  6.        0.      ]
 [      NaN       NaN       NaN  0.        0.01568]
 [      NaN       NaN       NaN       NaN       NaN]
 [      NaN       NaN       NaN       NaN       NaN]]
```

Listing 2.14: Output of Listing 2.13.

## 2.5. Reverse Mode

Besides the possibility to compute derivatives via univariate Taylor polynomial arithmetic, where all computations can be performed alongside the nominal computation, there is also the so-called *reverse mode*. The number of instructions to evaluate the gradient in the reverse mode is only a small constant multiple of the instructions required to evaluate the function itself. In particular this means that the time to evaluate the gradient is essentially independent of the number of arguments. The basic idea is to reverse the direction of the computation and accumulate derivatives starting at the dependent variable. The discussion in this section treats the most important aspects of the reverse mode. To keep the discussion concise it is referred to [Griewank and Walther, 2008] for a more detailed discussion.

One assumes that the underlying space is a Hilbert space with inner product $\langle \cdot | \cdot \rangle$. In textbooks the gradient $\nabla f(x)$ of a function $f : \mathbb{R}^N \to \mathbb{R}$ is defined such that

$$\langle \nabla f(x) | \dot{x} \rangle = \partial f(x; \dot{x})$$

holds. However, this definition lacks the generality to be useful in a computational context. In the AD theory one defines more generally the linear form

$$\langle \bar{y} | \partial F(x; \dot{x}) \rangle = \langle \bar{y} | \partial F(x) \dot{x} \rangle = \langle \partial F(x)^* \bar{y} | \dot{x} \rangle \ ,$$

where $\partial F(x)^*$ is the adjoint operator. In the following, only the case $F : \mathbb{R}^N \to \mathbb{R}^M$ is of interest. There, the adjoint of the matrix $\partial F(x)$ is simply the transposed matrix $\partial F(x)^T$. As one can see in the equation above, one goes back in the evaluation sequence. This action, at the level of "elementary" functions is called *pullback* in this thesis.

**Definition 2.5.1** (pullback)**.** Let $F : X \to Y$, $G : Y \to Z$ be two functions with $y = F(x)$ and $z = (G \circ F)(x)$. The action of going back one level of the functional dependence is called *pullback*. I.e.,

$$z = G(y) = (G \circ F)(x)$$

are two successive pullbacks. In the literature, the pullback is often written in star-notation

$$F^*(G)(y) := (G \circ F)(x) \ .$$

Let $F : U \subseteq X \to Y$ be a continuously differentiable mapping $x \mapsto y = F(x)$ between finite dimensional Hilbert spaces $X, Y$. As shown in Section 2.2, the derivative $\partial F(x; \dot{x}) \equiv \partial F(x)\dot{x}$ is a linear function in $\dot{x}$, i.e., an element of $\mathcal{L}(X, Y)$. Define the function $\bar{y} \in \mathcal{L}(Y, K)$ and let $\dot{y} = \partial F(x)\dot{x}$. Then one application of a pullback yields

$$\bar{y}(\dot{y}) = \bar{y}(\partial F(x)\dot{x}) = (\bar{y} \circ \partial F(x))(\dot{x}) =: \bar{x}(\dot{x}) \ . \tag{2.33}$$

In state space representation a pullback is written

$$s^{(2)} = \Phi_2(s^{(1)}) = (\Phi_2 \circ \Phi_1)(s^{(0)}) \ , \tag{2.34}$$

where $s^{(l)}$ is the state at step $l$ in the computation and $\Phi_l$ elementary transitions (c.f. Section 2.1) and hence one obtains for the pullback of the linear form

$$\bar{s}^{(l+1)}(\dot{s}^{(l+1)}) = \bar{s}^{(l+1)}(\partial \Phi_2(s^{(l)})\dot{s}^{(l)}) = \bar{s}^{(l)}(\dot{s}^{(l)}) \ . \tag{2.35}$$

There are $L$ elementary transitions to compute the overall function. If it is possible to bound the number of arithmetic operations to perform the pullback for each possible elementary transition by a constant $c$ then the total number of arithmetic operations to perform the overall pullback is bounded by $cL$.

### 2.5.1. Standard Pullback and Cheap Gradient Principle

The most important case in practice are the spaces $X = \mathbb{R}^N$, $Y = \mathbb{R}^M$ and $K = \mathbb{R}$ with the canonical inner product $\langle a | b \rangle = a^T b = \sum_{m=1}^M a_m b_m$. The standard linear form in AD theory is the $\mathbb{R}$-linear form

$$\bar{y}(\dot{y}) := \langle \bar{y} | \dot{y} \rangle = \bar{y}^T \dot{y} = (\bar{y}_1, \ldots, \bar{y}_M) \begin{pmatrix} \dot{y}_1 \\ \vdots \\ \dot{y}_M \end{pmatrix} = \sum_{m=1}^M \bar{y}_m \dot{y}_m \in \mathbb{R} \ ,$$

where $\bar{y}_m, \dot{y}_m \in \mathbb{R}$. I.e., the symbol $\bar{y}$ is overloaded and means either a finite dimensional vector $\bar{y} \in \mathbb{R}^M$ or the functional. A pullback can be written in coordinates as

$$\langle \bar{y} | \dot{y} \rangle = \bar{y}^T \partial F(x) \dot{x} = (\partial F(x)^T \bar{y})^T \dot{x} = \langle \partial F(x)^T \bar{y} | \dot{x} \rangle =: \langle \bar{x} | \dot{x} \rangle \ . \tag{2.36}$$

To give an explicit example, consider the task of computing the gradient $\nabla f(x)$ of the function

$$\begin{aligned} f : \mathbb{R}^2 &\to \mathbb{R} \\ x &\mapsto y = f(x) = \sin(x_1 + \cos(x_2) \cdot x_1) \ . \end{aligned}$$

Note that the symbol $f$ is used over $F$ since the function maps to the real numbers. One algorithm that evaluates $f$ is given by the three-part form, which has been copied from Example 2.1.1 for convenience.

| | | | | | |
|---|---|---|---|---|---|
| independent | $v_{-1}$ | $=$ | $x_1$ | $=$ | $3$ |
| independent | $v_0$ | $=$ | $x_2$ | $=$ | $7$ |
| | $v_1$ | $=$ | $\phi_1(v_0)$ | $=$ | $\cos(v_0)$ |
| | $v_2$ | $=$ | $\phi_2(v_1, v_{-1})$ | $=$ | $v_1 v_{-1}$ |
| | $v_3$ | $=$ | $\phi_3(v_{-1}, v_2)$ | $=$ | $v_{-1} + v_2$ |
| | $v_4$ | $=$ | $\phi_4(v_3)$ | $=$ | $\sin(v_3)$ |
| dependent | $y$ | $=$ | $v_4$ | | |

The gradient can then be computed by successive pullbacks of the linear form $\langle 1, \partial f(x; \dot{x}) \rangle = \nabla f(x)^T \dot{x}$:

$$\begin{aligned} \bar{y}^T \dot{y} &= \bar{y}^T \ \partial \phi_4(z)|_{z=v_3} \ \dot{v}_3 = \underbrace{\cos(v_3)}_{=\bar{v}_3} \dot{v}_3 \\ &= \bar{v}_3 \partial \phi_3(v_{-1}, v_2) \begin{pmatrix} \dot{v}_{-1} \\ \dot{v}_2 \end{pmatrix} = \underbrace{\bar{v}_3}_{=\bar{v}_{-1}} \dot{v}_{-1} + \underbrace{\bar{v}_3}_{=\bar{v}_2} \dot{v}_2 \\ &= \underbrace{(\bar{v}_{-1} + \bar{v}_2 v_1)}_{=\bar{v}_{-1}} \dot{v}_{-1} + \underbrace{\bar{v}_2 v_{-1}}_{=\bar{v}_1} \dot{v}_1 \\ &= \bar{v}_{-1} \dot{v}_{-1} + \underbrace{(-\bar{v}_1 \sin(v_0))}_{=\bar{v}_0} \dot{v}_0 \end{aligned} \ .$$

The interpretation is that the *bar values* are the wanted elements of the gradient

$$\bar{v}_{-1} \equiv \frac{\partial f}{\partial x_1} \quad \text{and} \quad \bar{v}_0 \equiv \frac{\partial f}{\partial x_2} \ .$$

It is important to note that one needs to **store** $v_0, v_1, v_3, v_4$ during the normal function evaluation since they are necessary in the reverse mode. Furthermore, notice that though symbolic expressions of the bar values are shown, they are in fact resolved instantly to numerical values by an AD tool. In other words, the bar values $\bar{v}_l$ are floating point numbers and not expressions. I.e., one traverses the computational graph in reverse direction and applies at at each node $\phi_l(v_{j \prec l})$ a pullback of the linear form

$$\bar{v}_l^T \dot{v}_l = \sum_{j \prec l} \bar{v}_l^T \frac{\partial \phi_l}{\partial v_j} \dot{v}_j \ .$$

That means one has to add $\bar{v}_l^T \frac{\partial \phi_l}{\partial v_j}$ into the bar variables $\bar{v}_j$. This incrementation has to be repeated when $v_j$ is argument of other functions. The extended three-part form

| | | | | |
|---|---|---|---|---|
| independent | $v_{n-N}$ | $=$ | $x_n$ | $n = 1, \dots, N$ |
| intermediates | $v_l$ | $=$ | $\phi_l(v_{j \prec l})$ | $l = 1, \dots, L$ |
| dependent | $y_{M-m}$ | $=$ | $v_{L-m}$ | $m = M-1, \dots, 0$ |
| init bar values | $\bar{v}_l$ | $=$ | $0$ | $l = 1-N, \dots, L$ |
| dependent | $\bar{v}_{L-m}$ | $=$ | $\bar{y}_{M-m}$ | $m = 0, \dots, M-1$ |
| | $\bar{v}_j^T$ | $+=$ | $\bar{v}_l^T \frac{\partial \phi_l}{\partial v_j}(v_{i \prec l})$ for $j \prec l$ | $l = L, \dots, 1$ |
| independent | $\bar{x}_n$ | $+=$ | $\bar{v}_{n-N}$ | $n = N, \dots, 1$ |

is therefore called *incremental adjoint recursion*. As one can see, it is necessary to compute quantities such as $\bar{v}_l^T \frac{\partial \phi_l}{\partial v_j}(v_{i \prec l})$. For the functions $\phi \in \{\pm, \cdot, \div, \sin, \dots\}$ as defined in *math.h* it is easy to derive pullback formulas:

- Let $z = f(x, y) = xy$, then the pullback reads

$$\bar{z}\dot{z} = \bar{z}\left(\frac{\partial f}{\partial x}(x, y)\dot{x} + \frac{\partial f}{\partial y}(x, y)\dot{y}\right) = \bar{z}y\dot{x} + \bar{z}x\dot{y} \; .$$

  That means to compute both $\bar{x} += \bar{z}y$ and $\bar{y} += \bar{z}x$ two multiplications are required.

- Let $z = f(x, y) = \frac{x}{y}$. The pullback formula is

$$\bar{z}\dot{z} = \frac{\bar{z}}{y}\dot{x} - \frac{\bar{z}z}{y}\dot{y} \; .$$

  In total, three arithmetic operations (neglecting the unary negation $-$) are necessary.

- Let $y = f(x) = \sin(x)$. It follows that the pullback

$$\bar{y}\dot{y} = \bar{y}\cos(x)\dot{x} = \bar{x}\dot{x}$$

  can be performed in two arithmetic instructions.

**Proposition 2.5.1** (cheap gradient principle)**.** *Let*

$$f : \mathbb{R}^N \to \mathbb{R}$$

*be continuously differentiable and given in three-part form using $L$ instructions as defined in the C-header* math.h*. Then it is possible to compute the gradient*

$$\nabla_x f(x) \in \mathbb{R}^N$$

*in less than $4L$ instructions from* math.h*.*

*Proof.* At first, the function has to be evaluated and all intermediate results have to be stored. This already requires $L$ arithmetic operations. One can show for all elementary functions in *math.h* that the algorithm to perform a pullback as given in (2.35) requires no more than three elementary function evaluations. Since the total number of instructions is $L$ it directly follows that $4L$ is an upper bound. This bound can be improved if the intermediate values are already computed. □

In practice, this result does not directly translate to

$$\mathrm{TIME}(\nabla f(x)) \leq 4\mathrm{TIME}(f(x)) \; .$$

The reason is the algorithmic challenge to provide the intermediate results. The traditional method is to store them in memory, possibly in combination with a checkpointing technique.

Since fetching data from memory is much slower than the processing speed it is often the case that it is better to recompute certain parts than fetching from memory. For some special functions it is possible to derive algorithms that do not require all intermediate results. Several important cases are treated in Chapter 3.

**Example 2.5.1** (Functions with Side Effects)**.** The state space representation is particularly useful to understand functions with side effects. Consider the program

```
s = zeros(2)
x = 1
s[0] = x
s[1] = s[0]
```

The question is what the corresponding adjoint program looks like. For that case, a state $s \in \mathbb{R}^2$ suffices. There are two states $s^{(1)} = (1,0)^T$ and $s^{(2)} = (1,1)^T$ and therefore the state transition is modeled by

$$s^{(2)} = f(s^{(1)}) = \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix} s^{(1)} .$$

Thus, in the reverse mode one computes

$$\bar{s}^{(2)T}\dot{s}^{(2)} = \bar{s}^{(2)T}\frac{\partial f}{\partial s}\dot{s}^{(1)} = \bar{s}^{(2)T}\begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix}\dot{s}^{(1)} = \begin{pmatrix} \bar{s}_1^{(2)} + \bar{s}_2^{(2)}, & 0 \end{pmatrix}\dot{s}^{(1)} = \bar{s}^{(1)T}\dot{s}^{(1)} .$$

## 2.5.2. Pullback in Univariate Taylor Polynomial Arithmetic

The standard pullback from Chapter 2.5.1 provides a method to compute the gradient of a real-valued function in an efficient way. In Chapter 2.4 it has been discussed how univariate Taylor polynomial arithmetic can be used to evaluate first and higher-order derivative tensors. The idea was to use univariate Taylor polynomial arithmetic (Chapter 2.3) in conjunction with interpolation/polarization identities. Now, the results from Chapter 2.3.5, where differentiability aspects were investigated, are used to define two different linear forms. As discussed in Chapter 2.3.5 one can regard extended functions $E_D(f) : \mathbb{R}[T]/(T^D) \to \mathbb{R}[T]/(T^D)$ as mappings between $D$ dimensional vector spaces. Explicitly, each output coefficient is a function of coefficients

$$\begin{aligned} y_{[0]} &= y_{[0]}(x_{[0]}) \\ y_{[1]} &= y_{[1]}(x_{[0]}, x_{[1]}) \\ \dots &= \dots \\ y_{[D-1]} &= y_{[D-1]}(x_{[0]}, \dots, x_{[D-1]}) . \end{aligned}$$

and the Jacobi×vector product reads in Toeplitz-form

$$\begin{pmatrix} \dot{y}_{[0]} & & & & \\ \dot{y}_{[1]} & \dot{y}_{[0]} & & & \\ \dot{y}_{[2]} & \dot{y}_{[1]} & \dot{y}_{[0]} & & \\ \vdots & \ddots & \ddots & \ddots & \\ \dot{y}_{[D-1]} & \cdots & \dot{y}_{[2]} & \dot{y}_{[1]} & \dot{y}_{[0]} \end{pmatrix} = \begin{pmatrix} J_{[0]} & & & & \\ J_{[1]} & J_{[0]} & & & \\ J_{[2]} & J_{[1]} & J_{[0]} & & \\ \vdots & \ddots & \ddots & \ddots & \\ J_{[D-1]} & \cdots & J_{[2]} & J_{[1]} & J_{[0]} \end{pmatrix} \begin{pmatrix} \dot{x}_{[0]} & & & & \\ \dot{x}_{[1]} & \dot{x}_{[0]} & & & \\ \dot{x}_{[2]} & \dot{x}_{[1]} & \dot{x}_{[0]} & & \\ \vdots & \ddots & \ddots & \ddots & \\ \dot{x}_{[D-1]} & \cdots & \dot{x}_{[2]} & \dot{x}_{[1]} & \dot{x}_{[0]} \end{pmatrix} .$$

Equivalently one can write the multiplications also as a matrix × vector product

$$\begin{pmatrix} \dot{y}_{[0]} \\ \dot{y}_{[1]} \\ \dot{y}_{[2]} \\ \vdots \\ \dot{y}_{[D-1]} \end{pmatrix} = \begin{pmatrix} J_{[0]} & & & & \\ J_{[1]} & J_{[0]} & & & \\ J_{[2]} & J_{[1]} & J_{[0]} & & \\ \vdots & \ddots & \ddots & \ddots & \\ J_{[D-1]} & \cdots & J_{[2]} & J_{[1]} & J_{[0]} \end{pmatrix} \begin{pmatrix} \dot{x}_{[0]} \\ \dot{x}_{[1]} \\ \dot{x}_{[2]} \\ \vdots \\ \dot{x}_{[D-1]} \end{pmatrix} .$$

Application of the standard pullback means to map $(\dot{y}_{[0]}, \ldots, \dot{y}_{[D-1]})^T$ to the real numbers, i.e., definition of a linear form and a pullback

$$(\bar{y}_{[0]}, \ldots, \bar{y}_{[D-1]}) \cdot \begin{pmatrix} \dot{y}_{[0]} \\ \vdots \\ \dot{y}_{[D-1]} \end{pmatrix} = (\bar{x}_{[0]}, \ldots, \bar{x}_{[D-1]}) \cdot \begin{pmatrix} \dot{x}_{[0]} \\ \vdots \\ \dot{x}_{[D-1]} \end{pmatrix} .$$

**Example 2.5.2.** The second derivative $f''(x_{[0]})$ of a function $f : \mathbb{R} \to \mathbb{R}$ is $\frac{\partial y_{[1]}}{\partial x_{[0]}} = J_{[1]}$, where $y_{[1]} = \frac{\partial}{\partial T} f(x_{[0]} + 1T)|_{T=0} = f'(x_{[0]})$. To extract this derivative one initializes $\bar{y}^T = (0, 1)$ and therefore obtains $\bar{x}^T = (J_{[1]}, J_{[0]})$.

**Lemma 2.5.2.** *Let*

$$\begin{aligned} F : \mathbb{R}^N &\to \mathbb{R}^M \\ x &\mapsto y = F(x) \end{aligned}$$

*be a $D$ times continuously differentiable function in an open neighborhood $U$ of $x$, $[y]_D = E_D(F)([x]_D)$ and $[\dot{y}]_D = E_D(\partial F)([x]_D)[\dot{x}]_D$. Consider the $\mathbb{R}$-linear form*

$$\tilde{y}([\dot{y}]_D) := \langle [\tilde{y}]_D | [\dot{y}]_D \rangle = \sum_{d=0}^{D-1} \tilde{y}_{[d]}^T \dot{y}_{[d]} = \sum_{d=0}^{D-1} \sum_{m=1}^{M} \tilde{y}_{[d],m} \dot{y}_{[d],m} , \tag{2.37}$$

*i.e., linear in each coefficient $\dot{y}_{[d],m}$, $d = 0, \ldots, D-1$ and $m = 1, \ldots, M$. Then the following expression holds*

$$\langle [\tilde{y}]_D | [\dot{y}]_D \rangle = \langle [\tilde{x}]_D | [\dot{x}]_D \rangle , \tag{2.38}$$

*where*

$$\tilde{x}_{[d]}^T = \sum_{0 \le i \le D-1-d} \tilde{y}_{[i+d]}^T J_{[i]} \quad \text{for } d = 0, \ldots, D-1$$

*with $[J]_D := E_D(\partial F)([x]_D)$.*

*Proof.* Using $\langle [\tilde{y}]_D | [\dot{y}]_D \rangle = \langle [\tilde{y}]_D | E_D(\partial F)([x]_D)[\dot{x}]_D \rangle$ and $[J]_D := E_D(\partial F)([x]_D)$, $[v]_D := [\dot{x}]_D$ resp. $[w]_D : \overset{D}{=} [J]_D [v]_D$ one finds that

$$\begin{aligned} \tilde{y}([\dot{y}]_D) = \sum_{d=0}^{D-1} \tilde{y}_{[d]}^T w_{[d]} &= \sum_{d=0}^{D-1} \tilde{y}_{[d]}^T \sum_{i+j=d} J_{[i]} v_{[j]} = \sum_{d=0}^{D-1} \sum_{i+j=d} \tilde{y}_{[d]}^T J_{[i]} v_{[j]} \\ &= \sum_{i+j \le D-1} \tilde{y}_{i+j}^T J_{[i]} v_{[j]} = \sum_{d=0}^{D-1} \underbrace{\left( \sum_{i \le D-1-d} \tilde{y}_{[i+d]}^T J_{[i]} \right)}_{=: \tilde{x}_{[d]}^T} v_{[d]} \\ &= \langle [\tilde{x}]_D | [\dot{x}]_D \rangle \end{aligned}$$

holds. □

There is also another good choice of a linear form. It's big advantage is that it means that the standard pullbacks from Chapter 2.5.1 can be evaluated in Taylor polynomial arithmetic. I.e., existing algorithms for univariate Taylor polynomial arithmetic can be reused. This is how the software ADOL-C [Griewank et al., 1999], Taylorpoly [Walter, 2010] and AlgoPy [Walter, 2009] implement the reverse mode.

**Lemma 2.5.3.** *Let the function*

$$\begin{aligned} F : \mathbb{R}^N &\to \mathbb{R}^M \\ x &\mapsto y = F(x) \end{aligned}$$

*be $D$ times continuously differentiable on an open neighborhood $U$ of $x$. Additionally, let $[y]_D = E_D(F)([x]_D)$ and $[\dot{y}]_D = E_D(\partial F)([x]_D)[\dot{x}]_D$ and consider the $\mathbb{R}[T]/(T^D)$-linear form*

$$\bar{y}([\dot{y}]_D) := [\bar{y}]_D^T [\dot{y}]_D \in \mathbb{R}[T]/(T^D) \,,$$

*where $[\bar{y}]_D \in \mathbb{R}[T]/(T^D)^M$ and $[\bar{y}]_D^T[\dot{y}]_D = \sum_{m=1}^M [\bar{y}_m]_D[\dot{y}_m]$. Then the following expression holds*

$$[\bar{y}]_D^T[\dot{y}]_D = [\bar{y}]_D^T \left(\partial(E_D(f)([x]_D))\{[\dot{x}]_D\}\right) = [\bar{y}]_D^T E_D(\partial f)([x]_D)[\dot{x}] =: [\bar{x}]_D^T[\dot{x}]_D \,. \tag{2.39}$$

*Proof.* The second equality follows from Corollary 2.3.8 (c.f. Chapter 2.3.5) and the associativity of polynomials. □

One can interpret this Lemma as follows: If $[\bar{y}]_D = w \in \mathbb{R}^M$ then $[\bar{x}]_D^T = E_D(w^T \frac{\partial F}{\partial x})([x]_D)$. Setting $w = e_i$ a Cartesian basis vector would yield the Taylor expansion of the $i$'th row of the Jacobian. Interpretation of the Taylor coefficients as derivatives yields higher-order derivatives. If $M = 1$ and $w = 1$ one obtains the Taylor expansion of the gradient $[\bar{x}]_D = E_D(\nabla f)([x]_D)$. E.g., propagating the UTP $[x]_2 = x_{[0]} + x_{[1]}T$ would yield $[\bar{x}]_2 = \bar{x}_{[0]} + \bar{x}_{[1]}T$ where $\bar{x}_{[0]} = \nabla_x f(x)$ and $\bar{x}_{[1]} = \nabla_x^2 f(x_{[0]}) \cdot x_{[1]}$, i.e., a Hessian-vector product.

Due to the structure of the Jacobian $E_D(\partial f)([x]_D)$ it is possible to show that the two forms are essentially equivalent. By that it is meant that the results of their pullbacks can be transformed into each other.

**Proposition 2.5.4.** *Let the function*

$$\begin{aligned} F : \mathbb{R}^N &\to \mathbb{R}^M \\ x &\mapsto y = F(x) \end{aligned}$$

*be $D$ times continuously differentiable on an open neighborhood $U$ of $x$. Furthermore, let $[y]_D = E_D(F)([x]_D)$ and $[\dot{y}]_D = E_D(\partial F)([x]_D)[\dot{x}]_D$ and consider the two pullbacks*

$$\sum_{d=0}^{D-1} \tilde{y}_{[d]}^T \dot{y}_{[d]} = \sum_{d=0}^{D-1} \tilde{x}_{[d]}^T \dot{x}_{[d]} \quad \textit{resp.} \quad [\bar{y}]_D^T[\dot{y}]_D = [\bar{x}]_D^T[\dot{x}]_D \,.$$

*If*

$$\bar{y}_{[D-1-k]} = \tilde{y}_{[k]} \quad k = 0, \ldots, D-1$$

*then the equation*

$$\tilde{x}_{[k]} = \bar{x}_{[D-1-k]} \quad k = 0, \ldots, D-1$$

*holds.*

*Proof.*

$$\tilde{x}_{[k]}^T = \sum_{0 \le i \le D-1-k} \tilde{y}_{[i+k]}^T J_{[i]} = \sum_{0 \le i \le D-1-k} \bar{y}_{[D-1-i-k]}^T J_{[i]} = \bar{x}_{[D-1-k]}^T \,.$$

□

It is important to note that the assumptions in the above Proposition 2.5.4 are sometimes not satisfied. For instance, it may be necessary to evaluate nested derivatives of the form

$$\nabla_x f(x) = \nabla_x (g(x) \frac{\partial h}{\partial x}(x)) \ .$$

One can compute $\frac{\partial h}{\partial x}(x)$ in univariate Taylor polynomial arithmetic, then extract coefficients from the univariate Taylor polynomial and use this information to build the Jacobian matrix. This *extraction of Taylor coefficients* is an operation which is not in Toeplitz-form and hence cannot be written as polynomial multiplication: a requirement to use the pullback from Lemma 2.5.3.

**Example 2.5.3** (Reverse Mode of Jacobian computations from Univariate Taylor Polynomials)**.** Consider the task of computing the Jacobian $g(x) = \frac{\partial f}{\partial x}(x)$ of a function $f : \mathbb{R} \to \mathbb{R}$. In the forward mode one obtains the Jacobian by

$$g = \begin{pmatrix} 0 & 1 \end{pmatrix} \begin{pmatrix} y_{[0]} \\ y_{[1]} \end{pmatrix} \ .$$

Application of the reverse mode yields

$$\tilde{g} \begin{pmatrix} 0 & 1 \end{pmatrix} \begin{pmatrix} \dot{y}_{[0]} \\ \dot{y}_{[1]} \end{pmatrix} = \begin{pmatrix} 0 & \tilde{g} \end{pmatrix} \begin{pmatrix} \dot{y}_{[0]} \\ \dot{y}_{[1]} \end{pmatrix} = \begin{pmatrix} 0 & \tilde{g} \end{pmatrix} \begin{pmatrix} J_{[0]} & 0 \\ J_{[1]} & J_{[0]} \end{pmatrix} \begin{pmatrix} \dot{x}_{[0]} \\ \dot{x}_{[1]} \end{pmatrix} = \begin{pmatrix} \tilde{g} J_{[1]} & \tilde{g} J_{[0]} \end{pmatrix} \begin{pmatrix} \dot{x}_{[0]} \\ \dot{x}_{[1]} \end{pmatrix}$$

The interpretation is that

$$\frac{\partial g}{\partial x_{[0]}}(x_{[0]}) = J_{[1]} \quad \text{and} \quad \frac{\partial g}{\partial x_{[1]}}(x_{[0]}) = J_{[0]}$$

when $\tilde{g} = 1$. At this point one can use Proposition 2.5.4 and evaluate the pullback algorithms in univariate Taylor polynomial arithmetic.

**Example 2.5.4.** Consider the function

$$\begin{aligned} f : \mathbb{R}^2 &\to \mathbb{R} \\ x_1, x_2 &\mapsto \sin(x_1) x_2 \end{aligned}$$

and evaluate $\nabla_x \frac{\partial^2 f}{\partial x_1 \partial x_2}$ at $x = (3, 4)$. The idea is to evaluate $H(x) := \frac{\partial^2 f}{\partial x_1 \partial x_2}(x)$ using the combination of univariate Taylor polynomial arithmetic and the polarization identity (2.22). Since the resulting computation in univariate Taylor polynomial arithmetic requires the extraction of Taylor coefficients to build the desired derivative tensor, one has to use the linear form defined in Lemma 2.5.2. I.e., to compute $\nabla_x \frac{\partial^2 f}{\partial x_1 \partial x_2}$ one has to take the following steps:

1. To evaluate $z = H(x)$ one can propagate two univariate Taylor polynomials

$$\begin{aligned} [y_{[0]}, y_{[1;1]}, y_{[2;1]}] &= E_3(f)([x_{[0]}, \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \end{pmatrix}]) \\ \text{and} \quad [y_{[0]}, y_{[1;2]}, y_{[2;2]}] &= E_3(f)([x_{[0]}, \begin{pmatrix} 1 \\ -1 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \end{pmatrix}]) \end{aligned}$$

   which can be written in combined form as

$$[y_{[0]}, \{y_{[1;1]}, y_{[1;2]}\}, \{y_{[2;1]}, y_{[2;2]}\}] = E_3(f)([x_{[0]}, \{\begin{pmatrix} 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ -1 \end{pmatrix}\}, \{\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \end{pmatrix}\}]) \ .$$

Application of the polarization identity (2.22) requires to extract Taylor coefficients of several propagated directions. Writing the univariate Taylor polynomials as vectors this is done as follows

$$z := \frac{\partial^2 f}{\partial x_1 \partial x_2} = \frac{1}{2}(y_{[2;1]} - y_{[2;2]}) = \begin{pmatrix} 0 & 0 & \frac{1}{2} \end{pmatrix} \left( \begin{pmatrix} y_{[0;1]} \\ y_{[1;1]} \\ y_{[2;1]} \end{pmatrix} - \begin{pmatrix} y_{[0;2]} \\ y_{[1;2]} \\ y_{[2;2]} \end{pmatrix} \right) .$$

One can describe the computation of $H(x)$ in three-part form

| | | | | | |
|---|---|---|---|---|---|
| independent | $v_{-1}$ | $=$ | $x_1$ | $=$ | $3$ |
| independent | $v_0$ | $=$ | $x_2$ | $=$ | $4$ |
| | $[v_1]_3$ | $=$ | $\phi_1(v_{-1})$ | $=$ | $v_{-1} + \{1,1\}T + \{0,0\}T^2$ |
| | $[v_2]_3$ | $=$ | $\phi_2(v_0)$ | $=$ | $v_0 + \{1,-1\}T + \{0,0\}T^2$ |
| | $[v_3]_3$ | $=$ | $\phi_3([v_1]_3)$ | $=$ | $E_3(\sin)([v_1]_3)$ |
| | $[v_4]_3$ | $=$ | $\phi_4([v_3]_3, [v_2]_3)$ | $=$ | $E_3(\text{mul})([v_3]_3, [v_2]_3)$ |
| | $v_5$ | $=$ | $\phi_5([v_4]_3)$ | $=$ | $v_{4;[2;1]}$ |
| | $v_6$ | $=$ | $\phi_6([v_4]_3)$ | $=$ | $v_{4;[2;2]}$ |
| | $v_7$ | $=$ | $\phi_7(v_{-1}, v_5)$ | $=$ | $\text{sub}(v_5, v_6)$ |
| | $v_8$ | $=$ | $\phi_8(v_0, v_6)$ | $=$ | $\text{mul}(\frac{1}{2}, v_7)$ |
| dependent | $y$ | $=$ | $v_4$ | | |

2. Now, in the reverse mode one would like to compute the sensitivity of $f(x)$ w.r.t. $x$. I.e., one performs successive pullbacks of the form

$$\begin{aligned} \bar{z}\dot{z} =& \bar{z}\frac{1}{2}(\dot{y}_{[2;1]} - \dot{y}_{[2;2]}) \\ =& \bar{z}\begin{pmatrix} 0 & 0 & \frac{1}{2} \end{pmatrix} \left( \begin{pmatrix} \dot{y}_{[0;1]} \\ \dot{y}_{[1;1]} \\ \dot{y}_{[2;1]} \end{pmatrix} - \begin{pmatrix} \dot{y}_{[0;2]} \\ \dot{y}_{[1;2]} \\ \dot{y}_{[2;2]} \end{pmatrix} \right) \\ =& \begin{pmatrix} 0 & 0 & \frac{1}{2}\bar{z} \end{pmatrix} \begin{pmatrix} \dot{y}_{[0;1]} \\ \dot{y}_{[1;1]} \\ \dot{y}_{[2;1]} \end{pmatrix} + \begin{pmatrix} 0 & 0 & -\frac{1}{2}\bar{z} \end{pmatrix} \begin{pmatrix} \dot{y}_{[0;2]} \\ \dot{y}_{[1;2]} \\ \dot{y}_{[2;2]} \end{pmatrix} . \end{aligned}$$

From there, one could pullback the form until one obtains the result $\bar{x}_{[0]}$. In practice, AD tools like ADOL-C and AlgoPy use the linear from from Lemma 2.5.3. If one sorts the coefficients in reverse order as described by Proposition 2.5.4 one can use such software to compute the remaining pullbacks. I.e., in a reverse mode one has to set

$$[\bar{y}]_3 = [\{\frac{1}{2}\bar{z}, -\frac{1}{2}\bar{z}\}, 0, 0]_3 , \tag{2.40}$$

i.e., initialize the zeroth coefficient and not the last one. Then one finds the desired gradient is $\nabla_x f(x) = \bar{x}_{[2;1]} + \bar{x}_{[2;2]}$. Listing 2.15 shows a complete example that illustrates how existing AD tools can be used in combination with the results shown in this section.

```
import algopy, numpy

def eval_f(x):
    return algopy.sin(x[0]) * x[1]


# STEP 1: Taylor polynomial arithmetic
#         to compute d^2f/dx_1 dx_2

x = algopy.UTPM(numpy.zeros((3,2,2)))
x.data[0] = [3,4]
x.data[1,:,:] = [[1, 1],[1,-1]]
```

```
cg = algopy.CGraph()
x = algopy.Function(x)
tmp = eval_f(x)
cg.trace_off()
cg.independentFunctionList = [x]
cg.dependentFunctionList = [tmp]

# apply polarization
y = 0.5*(tmp.x.data[2,0] - tmp.x.data[2,1])

# STEP 2: pullback
tmpbar = algopy.zeros(tmp.x.shape, dtype=tmp.x)
tmpbar.data[0,0] = 0.5
tmpbar.data[0,1] = -0.5
cg.pullback([tmpbar])

xbar = x.xbar.data[2,0] + x.xbar.data[2,1]

print 'AD solution of y   =', y
print 'Symbolic       y   =', algopy.cos(3)
print 'AD solution of xbar=', xbar
print 'Symbolic       xbar=', [-algopy.sin(3),0]
print 'difference         =', xbar - [-algopy.sin(3),0]
```

Listing 2.15: The program computes $\nabla_x \frac{\partial^2 f}{\partial x_1 \partial x_2}$ at $x = (3, 4)$ of $f : x_1, x_2 \mapsto \sin(x_1)x_2$ using a polarization identity and the reverse mode. The solution is compared to the symbolically derived solution.

```
AD solution of y   = -0.9899924966
Symbolic       y   = -0.9899924966
AD solution of xbar= [-0.14112001  0.        ]
Symbolic       xbar= [-0.14112000805986721, 0]
difference         = [ -2.77555756e-17   0.00000000e+00]
```

Listing 2.16: Output of Listing 2.15.

*Remark* (which linear form?). The advantage of using the linear from Lemma 2.5.3 is that one obtains the derivative of all coefficients $y_{[d]}$ w.r.t. the zeroth coefficient $x_{[0]}$. As discussed in Chapter 2.4 one propagates several directions at once. The zeroth coefficient is shared by all directions. However, one can see in (2.40) it may be necessary that there are several "copies" of the zeroth coefficient. In consequence, existing algorithms for vectorized univariate Taylor polynomial arithmetic cannot directly be used.

## 2.6. Nesting Derivatives

Besides the interpolation approach (resp. multivariate Taylor polynomial arithmetic) it is also possible to compute higher-order partial derivatives by *nesting derivatives*. The idea is to generate a computational graph which allows the computation of the desired first-order derivative. For instance, to compute the gradient one can first compute the function itself and store all intermediate values. In a second step the reverse mode goes through this computational graph and records all operations that occur during the reverse mode. As result one obtains a new, larger computational graph which computes $(y, \bar{x}) = \bar{F}(x, \bar{y})$. A brief discussion how operator overloading techniques can be used for such an approach is discussed by Christianson [1993]. Similarly, it is also possible to trace the operations occurring during the forward mode. One obtains an algorithm in three-part form that can at this point be differentiated once more either in the forward or reverse mode. Bell calls this approach *multi-level taping* and has implemented it in CppAD [Bell, 2010]. Consider for instance the function

$$\begin{aligned} f : \mathbb{R}^2 &\to \mathbb{R} \\ x &\mapsto y = f(x) = (x_1 x_2 + x_1)x_2 \end{aligned}$$

and let $\frac{\partial}{\partial x_2} f(x)$ be the desired derivative. For the computation of the derivative using univariate Taylor polynomial arithmetic the independent variables are initialized as $[x_1]_2 = x_{1;[0]} + 0T$ and $[x_2]_2 = x_{2;[0]} + 1T$, where $x_{1;[0]}$ and $x_{2;[0]}$ are the independent variables. Tracing the operations of the univariate Taylor polynomial arithmetic one obtains the computational graph of $g(x) = \frac{\partial}{\partial T}(x + \binom{0}{1}T)\Big|_{T=0}$. Both graphs are shown in Figure 2.9.

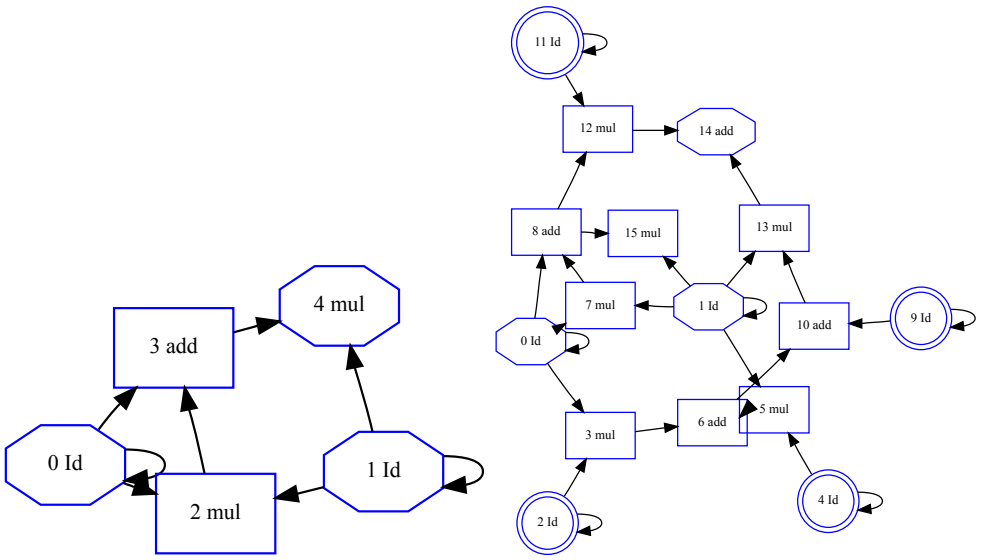



Figure 2.9.: On the left the computational graph of $f(x_1, x_2) = (x_1 x_2 + x_1)x_2$ is shown. On the right one can see the computational graph when the function $f(x_1, x_2)$ is evaluated in first-order Taylor polynomial arithmetic. The hexagons depict independent and dependent variables.

Unfortunately, especially in mixed language settings, it may not be possible to trace the evaluation of $E_D(f)([x]_2)$. For instance, ADOL-C [Griewank et al., 1999] or SolvIND [Albersmeyer and Kirches, 2007–] can both be used for univariate Taylor arithmetic as well as a reverse mode but are a black box otherwise. Then it is necessary to use techniques as described above.

## 2.7. Alternatives

Next to AD techniques there exist several other possibilities to evaluate derivatives: symbolic differentiation (SD), finite differences (FD) and the complex step derivative approximation (CSDA). None of the methods is per se better than the others and one can construct examples and use cases where one method excels and the others may even fail. The purpose of this section is to investigate the differences between the methods and highlight in particular their issues.

### 2.7.1. Symbolic Differentiation

Mathematical formulas are readily available in symbolic form. It is possible to chain symbolic expressions together and manipulate them. A special kind of symbolic computation is the application of the chain rule to a given expression. There are two types of issues that appear in symbolic differentiation.

1. The first issue concerns symbolic computations. As it has been observed many times, symbolic expressions often grow exponentially fast, especially if there is some kind of

recursion. Consider, for instance, the recursion

$$\begin{pmatrix} x^+ \\ v^+ \end{pmatrix} = F(x,v) = \begin{pmatrix} x + \left( \sqrt{\left(\frac{x^T v}{\|v\|^2}\right)^2 - \frac{\|x\|^2 - 1}{\|v\|^2}} - \frac{x^T v}{\|v\|^2} \right) v \\ P(x^+)v \end{pmatrix} ,$$

where $P(x^+) = \mathbf{1} - 2\frac{ww^T}{\|w\|^2}$, $w = w(x^+) = 2x^+$ and $g(x) = \|x\|^2 - 1 = x_1^2 + x_2^2 - 1$. It describes the reflection points $x$ and directions $v$ or a ray of light propagating in a cylindric mirror. The next reflection point $x^+$ and direction $v^+$ are computed by $x^+, v^+ = F(x,v)$. Written as expressions that contain only scalar valued symbols, one obtains

$$\begin{aligned} x^+ &= \begin{pmatrix} x_1 + v_1 g(x,v) \\ x_2 + v_2 g(x,v) \end{pmatrix} \\ v^+ &= \begin{pmatrix} v_1 - \frac{(2x_1 + 2v_1 g(x,v))(v_1(x_1 + v_1 g(x,v)) + v_2(x_2 + v_2 g(x,v)))}{(x_1 + v_1 g(x,v))^2 + (x_2 + v_2 g(x,v))^2} \\ v_2 - \frac{(2x_2 + 2v_2 g(x,v))(v_1(x_1 + v_1 g(x,v)) + v_2(x_2 + v_2 g(x,v)))}{(x_1 + v_1 g(x,v))^2 + (x_2 + v_2 g(x,v))^2} \end{pmatrix} \end{aligned}$$

where the common subexpressions

$$f(x,v) := \sqrt{\frac{1.0 - x_1^2 - x_2^2}{v_1^2 + v_2^2} + \frac{(v_1 x_1 + v_2 x_2)^2}{(v_1^2 + v_2^2)^2}}$$

and

$$g(x,v) = \left( -\frac{v_1 x_1 + v_2 x_2}{v_1^2 + v_2^2} + f(x,v) \right)$$

have already been identified and substituted. To compute the next reflection point and direction, the symbolic expression for $x^+, v^+$ has to be substituted in $x^+, v^+ = F(x,v)|_{x=x^+, v=v^+}$. Using SymPy (see SymPy Development Team [2009]) this already takes several minutes on a 2GHz computer and would take many pages of this document. For this reason, purely symbolic computations are generally discouraged. A more detailed discussion of the ray tracing problem is given in Appendix D.3.

2. The second issue is directly related to symbolic differentiation itself. Again, this is most conveniently illustrated by an example. Consider the function

$$f(x) = x(x + yx) .$$

This expression corresponds to a directed acyclic graph (the computational graph). To evaluate the function one has to translate the computational graph into a sequence of instructions. One possible sequence is

$$\begin{aligned} v_1 &= yx \\ v_2 &= x + v_1 \\ v_3 &= x v_2 . \end{aligned}$$

One can now apply the reverse mode symbolically, i.e.,

$$\begin{aligned} \mathrm{d}v_3 &= \mathrm{d}(xv_2) = \mathrm{d}x v_2 + x\mathrm{d}v_2 \\ &= \mathrm{d}x v_2 + x\mathrm{d}(x + v_1) = \mathrm{d}x v_2 + x\mathrm{d}x + x\mathrm{d}v_1 \\ &= (v_2 + x)\mathrm{d}x + x\mathrm{d}(yx) = (v_2 + x)\mathrm{d}x + xy\mathrm{d}x + x^2\mathrm{d}y \\ &= (v_2 + x + xy)\mathrm{d}x + x^2\mathrm{d}y . \end{aligned}$$

Ergo, the symbolic gradient is

$$\nabla f(x, y) = (v_2 + x + xy, x^2) \ .$$

As one can see, it consists of two expressions.

Assume for now that all intermediate values ($v_2$ in the above example) are given as expressions of the independent variables. Each element of the gradient is a sum over all paths products from independent to dependent variables. The path product is the product of all partial derivatives that are associated to the edges of the graph. I.e., the expression of one element of the gradient can be much larger than the function's expression graph. Consider a function with $10^6$ instructions and $10^4$ independent variables. Hence, symbolic differentiation requires memory for $10^4$ potentially large expressions. Furthermore, it is necessary to compile $10^4$ expressions which can take a while and leads to large binary files. And finally, each expression has to be evaluated separately. I.e., the computation of the gradient can be $10^4$ times as expensive as the function itself. For a discussion how to optimize the evaluation of symbolic derivatives see for instance Guenter [2007].

### 2.7.2. Numerical Differentiation

In exact arithmetic, the finite differences scheme

$$\Delta_t[f](x; v) := \frac{f(x + vt) - f(x)}{t} \ .$$

can be used to approximate the true directional derivative of a function $f \in C^1(\mathbb{R})$ arbitrarily well by driving $t$ to zero. However, in finite precision arithmetic, all operations have an error. In particular, even if the inputs are exactly representable in finite precision, the output is typically not. The function evaluated in floating point arithmetic is denoted $\tilde{f}(x)$ and the difference between the true and floating point solution is denoted $\delta = \tilde{f}(x) - f(x)$. One obtains

$$\begin{aligned}
\Delta_t[\tilde{f}](x; v) =& \frac{\tilde{f}(x + tv) - \tilde{f}(x)}{t} = \frac{f(x + tv) + \delta_1 - f(x) + \delta_2}{t} \\
=& \frac{f(x + tv) - f(x)}{t} + \frac{\delta_1 + \delta_2}{t} \\
=& \partial f(x; v) - \frac{R(x; tv)}{t} + \frac{\delta_1 + \delta_2}{t} \ ,
\end{aligned}$$

where $R(x; tv) \in o(vt)$ is the remainder of the Taylor expansion (see Theorem 2.2.6). Hence, the difference between the true and the numerically computed directional derivative is

$$\Delta_t[\tilde{f}](x; v) - \partial f(x; v) = \underbrace{\frac{R(x; tv)}{t}}_{\overset{t\to 0}{\to} 0} - \underbrace{\frac{\delta_1 + \delta_2}{t}}_{\overset{t\to 0}{\to} \infty} \ .$$

One obtains a smaller finite differences error as $t$ goes to zero as long as the remainder $R(x; tv)$ dominates. However, if $t$ is chosen too small, the random errors dominate. For twice continuously differentiable functions $f \in C^2(\mathbb{R})$ the remainder can be written as $R(x; tv)/t = \frac{1}{2} f''(\xi)t$ and therefore the optimal step size is $t = \sqrt{\frac{2(\delta_1+\delta_2)}{f''(\xi)}}$.

Higher-order derivatives can be computed using the *forward difference* formula

$$\Delta_t^d[f](x;v) = \sum_{k=0}^{d} (-1)^k \binom{d}{k} f(x + (d-k)tv)$$

$$\text{and it holds that} \quad \frac{\partial^d f}{\partial x^d}(x) = \frac{\Delta_t^d[f](x;v)}{t^d} + \mathcal{O}(t) \ .$$

E.g., $f^{(2)}(x;v) = \frac{f(x+2tv)-2f(x+tv)+f(x)}{t^2} + \mathcal{O}(t)$ and $f^{(3)}(x;v) = \frac{f(x+3tv)-3f(x+2tv)+3f(x+tv)-f(x)}{t^3} + \mathcal{O}(t)$. One can see that a too small $t$ yields a large overall error, even when the finite precision error is close to the machine precision.

To obtain an optimal step size one has to expand each term $f(x+(d-k)tv)$ in a Taylor series expansion and collect the remainder terms. I.e., for sufficiently smooth functions one finds that the optimal step size is defined by

$$\min_{t \in \mathbb{R}} \left( \Delta_t^d[\tilde{f}](x;v) - f^{(d)}(x;v) \right) = \min_t \mathcal{O}(t) + \frac{\delta}{t^d}$$

and obtains approximately

$$t = \left( \frac{d\delta}{\mathcal{O}(1)} \right)^{\frac{1}{d+1}} \ .$$

Assuming that that $\mathcal{O}(1) \approx 1$ and $\delta \approx 10^{-16}$ yields for second-order derivatives the optimal step size $t = 10^{-16/3} \approx 4.6 \cdot 10^{-6}$ and for third-order derivatives $t = 10^{-16/4} = 10^{-4}$. This behavior can be seen in Figure 2.10. Since typically neither the magnitude of $\mathcal{O}(1)$ nor the finite precision error $\delta$ is known one has to perform numerical tests. As one can see in Figure 2.10 one can easily get "wrong" answers when the wrong step size $t$ is chosen, especially for higher-order derivatives.

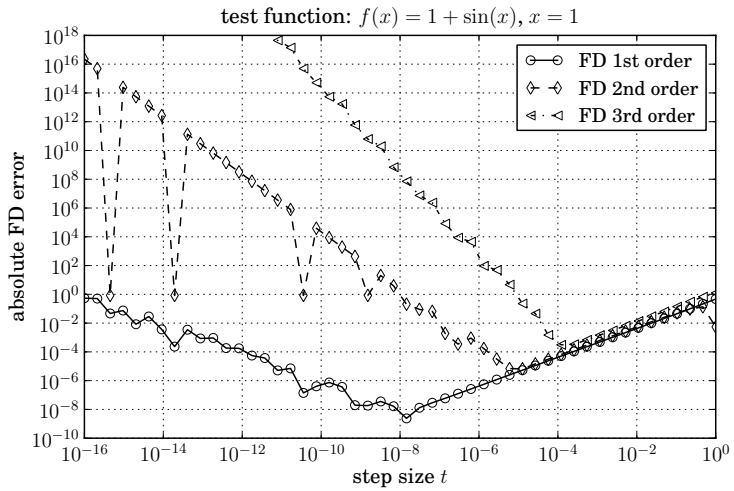



Figure 2.10.: This plot shows the absolute error between the symbolic and the FD solution, i.e., $|\frac{\partial^d}{\partial x^d} f(1) - \Delta_t^d[f](1)|$. It is assumed that the symbolic solution is correct up to machine precision EPS ($\approx 10^{-16} \approx 2^{-53}$ for IEEE 754 Float64). One can see that there is an optimal value for the step size $t \in \mathbb{R}$. Nonetheless, even the optimal $t$ for first-order derivatives results in an error of order $\sqrt{\text{EPS}} \approx 10^{-8}$. For higher-order derivatives, the optimal $t$ only gives rise to a rather inaccurate derivative approximation. Furtheremore, missing the optimal step size $t$ results in a large error. E.g., for third order derivatives the optimal $t$ is about $10^{-4}$. Since this is not known a priori, the wrong guess $t = 10^{-5}$ yields an error of order one.

### 2.7.3. Complex Step Derivative Approximation

Somewhat related to numerical differentiation by finite differences is the *complex step derivative approximation* (CSDA) which is also known as the *complex variable method*. The idea is to evaluate functions $f : \mathbb{R}^N \to \mathbb{R}$ in complex arithmetic to obtain accurate derivatives. More precisely, a Taylor approximation about some small imaginary number yields

$$f(x + ivt) = f(x) + \partial_x f(x) ivt - \frac{1}{2}\partial_x^2 f(x)\{v, v\}t^2 - i\frac{1}{6}\partial_x^3 f(x)\{v, v, v\}t^3 + \mathcal{O}(t^4) \, . \tag{2.41}$$

Hence, one obtains

$$\partial_x f(x)v = \frac{\Im(f(x + ivt))}{t} + \mathcal{O}(t^2) \tag{2.42}$$

if the imaginary part $\Im(f(x+ivt)) = \partial_x f(x)vt+\mathcal{O}(t^3)$ is divided by $t$. Newman et al. [1998] argue that the missing subtraction in the numerator results in a much better accuracy compared to finite differences, granted $t$ is chosen sufficiently small. For discussion where CSDA is compared to AD see Joaquim et al. [2001], Martins et al. [2003].

# 3. Evaluating Derivatives of Numerical Linear Algebra Functions

Many problems in scientific computing can be formulated as computer programs containing numerical linear algebra (NLA) routines. Consider for instance the following function

$$\begin{aligned} f : \mathbb{R}^{M\times N} \times \mathbb{R}^{K\times N} &\to \mathbb{R} \\ (J_1, J_2) &\mapsto y = \operatorname{tr}(C) \end{aligned} \tag{3.1}$$

$$\text{where} \quad C = C(J_1, J_2) = (\mathbf{I}, 0) \begin{pmatrix} J_1^T J_1 & J_2^T \\ J_2 & 0 \end{pmatrix}^{-1} \begin{pmatrix} \mathbf{I} \\ 0 \end{pmatrix} . \tag{3.2}$$

$\mathbf{I}$ is the identity matrix and $M \geq N$, $N \geq K$. To guarantee that the matrix inversion is well-defined it is assumed here that $\operatorname{rank}(J_2) = K$ and $\operatorname{rank}(J) = N$, where $J = (J_1^T, J_2^T)^T \in \mathbb{R}^{(M+K)\times N}$. The details are elaborated in Chapter 5.2.3, where it is also discussed that (3.2) can alternatively be written as

$$C = Q_2^T \left( Q_2 J_1^T J_1 Q_2^T \right)^{-1} Q_2 . \tag{3.3}$$

The matrix $Q_2$ results from a QR decomposition $J_2^T = (Q_1^T, Q_2^T)(L, 0)^T$. Körkel [2002] argues that, since $J_1$ can be ill-conditioned, one should avoid the matrix multiplication $J_1^T J_1$ as it would square the condition number. One possibility to avoid this multiplication is to perform another QR decomposition $J_1 Q_2^T = Q_3 R$ and to invert the triangular part $R \in \mathbb{R}^{(N-K)\times(N-K)}$ separately, so that finally

$$C = \left( R^{-T} Q_2 \right)^T \left( R^{-T} Q_2 \right) . \tag{3.4}$$

As discussed in Chapter 2.1, one can write the function evaluation of the above $f$ in *three-part form*

$$\begin{aligned} v_{n-N} &= J_n && n = 1, 2 \\ v_l &= \phi_l(v_{j\prec l}) && l = 1, \ldots, L \\ y &= v_L \end{aligned}$$

where $\phi_l$ are the *elementary functions*, $v_l$ the *intermediate values* and $v_{i\prec l}$ denotes the tuple of input arguments of $\phi_l$. What functions are regarded as elementary depends on the abstraction level. One possible sequence is shown in Table 3.1. There, the instruction "getitem" takes as second argument a slice index $(:, K+1:)$ which means that a submatrix is formed including all columns starting from $K+1$. $\operatorname{dot}(A, B) := AB$ is the usual matrix-matrix product and $X = \operatorname{solve}(A, B)$ is the functional dependence of the linear system $AX = B$. $\operatorname{tr}(A)$ is the trace of a matrix $A$ and $\operatorname{transpose}(A) = A^T$ is the transposition. $(Q, R) = \operatorname{qrfull}(A)$ computes a QR decomposition of $A \in \mathbb{R}^{M\times N}$, $M \geq N$, where $Q \in \mathbb{R}^{M\times M}$ and $R \in \mathbb{R}^{M\times N}$. Since only the upper $N \times N$ submatrix of $R$ is nonzero, it is also possible to write the QR decomposition as $\tilde{Q}, \tilde{R} = \operatorname{qr}(A)$, where now $\tilde{Q} \in \mathbb{R}^{M\times N}$ consists of the first $N$ columns of $Q$ and $\tilde{R} \in \mathbb{R}^{N\times N}$ is the nonzero part of $R$. One would like to have the possibility to evaluate derivatives of $f(J_1, J_2)$ based on the sequence given in Table 3.1. More explicitly, assume that one would like to evaluate

1. the gradient $\nabla f(J_1, J_2)$ in the forward and reverse mode
2. the Hessian $\nabla^2 f(J_1, J_2)$ in forward mode and combined forward/reverse mode

and potentially also higher-order derivatives.

| | | | | | |
|---|---|---|---|---|---|
| independent | $v_{-1}$ | $=$ | $J_1$ | | |
| independent | $v_0$ | $=$ | $J_2$ | | |
| | $v_1$ | $=$ | $\phi_1(v_0)$ | $=$ | $\text{transpose}(v_0)$ |
| | $(v_3, v_4)$ | $=$ | $\phi_2(v_0)$ | $=$ | $\text{qrfull}(v_1)$ |
| | $v_5$ | $=$ | $\phi_5(v_3)$ | $=$ | $\text{getitem}(v_3, (:, K+1:))$ |
| | $v_6$ | $=$ | $\phi_6(v_5)$ | $=$ | $\text{transpose}(v_5)$ |
| | $v_7$ | $=$ | $\phi_7(v_{-1}, v_5)$ | $=$ | $\text{dot}(v_{-1}, v_5)$ |
| | $(v_9, v_{10})$ | $=$ | $\phi_8(v_7)$ | $=$ | $\text{qr}(v_7)$ |
| | $v_{11}$ | $=$ | $\phi_{11}(v_{10})$ | $=$ | $\text{transpose}(v_{10})$ |
| | $v_{12}$ | $=$ | $\phi_{12}(v_{11}, v_6)$ | $=$ | $\text{solve}(v_{11}, v_6)$ |
| | $v_{13}$ | $=$ | $\phi_{13}(v_{12})$ | $=$ | $\text{transpose}(v_{12})$ |
| | $v_{14}$ | $=$ | $\phi_{14}(v_{12}, v_{13})$ | $=$ | $\text{dot}(v_{13}, v_{12})$ |
| | $v_{15}$ | $=$ | $\phi_{15}(v_{14})$ | $=$ | $\text{tr}(v_{14})$ |
| dependent | $y$ | $=$ | $v_{15}$ | | |

Table 3.1.: Sequence of instructions to compute the function $f(J_1, J_2)$ defined in (3.1).

The purpose of this section is to apply and generalize the framework from Chapter 2 to such matrix computations. There are several possibilities, each corresponding to another hierarchical level. Their pros and cons are discussed in Chapter 3.2. It is argued that working at the level of numerical linear algebra functions provides a good middle ground between the level of basic elementary functions and a fully symbolic treatment. After that it is shown in Chapter 3.3 how the results from Chapter 2 generalize to numerical linear algebra functions based on the notion of matrix calculus. In particular, it is shown how univariate Taylor polynomials can be propagated through NLA functions and it is shown how the reverse mode can be expressed in terms of matrix calculus. The following sections are then concerned with the application of the general principle to a variety of useful numerical linear algebra functions such as the Cholesky (Chapter 3.6), QR (Chapter 3.7 resp. Chapter 3.8) and the real symmetric eigenvalue decomposition (Chapter 3.9). The chapter concludes with an experimental runtime comparison in Chapter 3.11, where it is demonstrated that the application of structured AD can lead to significantly better runtimes.

## 3.1. Related Work and Scope

There exist several textbooks on matrix calculus, for instance by Magnus and Neudecker [1999], Seber [2007], Healy [2000], Schott [1997]. However, these books do not put any focus on the fact that matrix operations are typically only a small part of a larger computational procedure. I.e., neither the forward mode nor the reverse mode are mentioned. Giles [2007] collects several matrix calculus results and describes them in standard AD terminology. More explicitly, he applies first-order forward and reverse mode AD to matrix operations such as the matrix product, matrix inversion, etc. Beyond such first-order results, there are only very few publications where Taylor polynomial arithmetic is applied to matrix operations. For instance Phipps [2003] derives algorithms for univariate Taylor polynomial arithmetic of the matrix product, matrix inversion and solution of linear systems. In that view, the following discussion unifies many existing techniques in a more general framework which allows also the computation of

higher-order derivatives. The algorithms for the Cholesky, QR and real symmetric eigenvalue decomposition are novel and have interesting properties: The QR decomposition can be used to characterize "derivatives" of the nullspace of a matrix and the symmetric eigenvalue decomposition can be used to find Taylor series approximations of the eigenvalues also in the case when there are multiple eigenvalues. That means, despite the fact that the mapping is not (Fréchet-)differentiable, it is possible to compute directional derivatives.

## 3.2. A Case for Structured Algorithmic Differentiation

Traditionally, AD tools consider algorithms to be a sequence of basic elementary functions as defined in Chapter 2.3.6. I.e., the sequence consists of operations such as $\pm, \cdot, \div, \exp, \sin$, etc. The chain rule is applied at this hierarchical level. Thus, it is just necessary to apply symbolic differentiation on the level of basic elementary functions. It must be emphasized that one can include more elaborate "elementary" functions as long as it is possible to differentiate them analytically. Working at a higher hierarchical level allows one to consider more global information than a mechanical application of the chain rule. Consider for instance the QR factorization of a $\mathbb{R}^{2\times 2}$ matrix using Givens rotations. Tracing the evaluation results in a computational graph as shown in Figure 3.1. The size of the graph quickly gets larger since the QR decomposition requires $N^3$ arithmetic operations to factorize $\mathbb{R}^{N\times N}$ matrices. E.g., for a $\mathbb{R}^{100\times 100}$ matrix the graph would have about $10^6$ nodes. The sheer size of such a computational graph means that

- there will be much overhead in case operator overloading is used
- the memory requirement scales with $N^3$ in the reverse mode of AD if no additional care is taken
- it is not possible to use existing high-performance implementations.

In contrast, regarding the QR decomposition as an *atomic function* (in the sense of indivisible) results in a very small computational graph (Figure 3.2).

A lot of research and work has gone into the derivation of numerically stable NLA algorithms and their efficient implementation in software libraries as for instance LAPACK (c.f. Anderson et al. [1999]) and ATLAS (c.f. Whaley et al. [2001]). Such software has the advantage that it is verified and provides useful design patterns. The matrix factorizations typically make use of pivoting. Additionally, the code often takes different program branches to avoid unnecessary operations. Though it has been shown in [Griewank and Walther, 2008, Proposition 14.2] that the set of non-differentiable points due to program branches has measure zero, such events appear on a regular basis in practice. To give an explicit example, consider the BLAS function *dnrm2.f* of which the interesting part is shown in Listing 3.1. It computes the norm $\|x\|$ of a vector $x \in \mathbb{R}^N$ in a numerically stable way by scaling the vector $x$. In formulas

$$\|x\| = \sqrt{\sum_{n=1}^{N} x_n^2} = s\sqrt{\sum_{n=1}^{N} \left(\frac{x_n}{s}\right)^2},$$

where $s = \max_n |x_n|$. One can see in line 9 that there is an additional check whether the element $x_n$ is zero. In consequence, the function is not differentiable when one element of the vector vanishes. Clearly, this is not a border-line case and happens a lot in practice.

```
      IF (N.LT.1 .OR. INCX.LT.1) THEN
          NORM = ZERO
      ELSE IF (N.EQ.1) THEN
          NORM = ABS(X(1))
5     ELSE
          SCALE = ZERO
```

```
            SSQ = ONE
            DO 10 IX = 1,1 + (N-1)*INCX,INCX
                 IF (X(IX).NE.ZERO) THEN
                      ABSXI = ABS(X(IX))
                      IF (SCALE.LT.ABSXI) THEN
                           SSQ = ONE + SSQ* (SCALE/ABSXI)**2
                           SCALE = ABSXI
                      ELSE
                           SSQ = SSQ + (ABSXI/SCALE)**2
                      END IF
                 END IF
   10       CONTINUE
            NORM = SCALE*SQRT(SSQ)
         END IF
```

Listing 3.1: Part of the BLAS routine dnrm2.f.

Another example is Algorithm 3.2.1 which performs a Givens rotation. As one can see, the algorithm contains a check $b = 0$. At the core of the issue is that the control flow depends only on the nominal solution but not on the derivatives. I.e., if $b = 0$ then the three-part form of the algorithm is

$$v_{-1} = a, v_0 = b\,; \quad c = v_1 = 1\,, \quad s = v_2 = 0$$

As result, application of the reverse mode would yield the incorrect result $\bar{a} = 0$ and $\bar{b} = 0$. In

$s, c =$ **givens**$(a, b)$
**input** : $a, b \in \mathbb{R}$
**output**: $s, c \in \mathbb{R}$

**if** $b = 0$ **then**
  $c = 1$; $s = 0$
**else**
  **if** $|b| > |a|$ **then**
    $r = -\frac{a}{b}$; $s = \frac{1}{\sqrt{1+r^2}}$; $c = sr$
  **else**
    $r = -\frac{b}{a}$; $c = \frac{1}{\sqrt{1+r^2}}$; $s = cr$
  **end**
**end**
**return** $s, c$

**Algorithm 3.2.1:** Computes $c, s$ from $a, b$ using Givens rotation.)

conclusion one can say that one has to be very careful if one applies an automatic differentiation tool to such codes. That means, if it is desired to apply AD on an elementary level, it is necessary to reimplement many algorithms because existing implementations

- may use elaborate algorithms containing non-differentiable operations such as checks if certain matrix elements are zero or possibly look-up tables
- often use pivoting which introduces branches, a fact that makes it hard to handle this case using operator overloading.

Another complication are algorithms whose output is not unique, for instance the eigenvalue decomposition with repeated eigenvalues. In that case the algorithm does not contain all information of the function itself. It has been reported by Guerrieri that the AD tool TAPENADE fails to generate code that produces the correct derivatives. The problems arose in particular for the Schur and singular value decomposition. Their solution was to provide hand-made

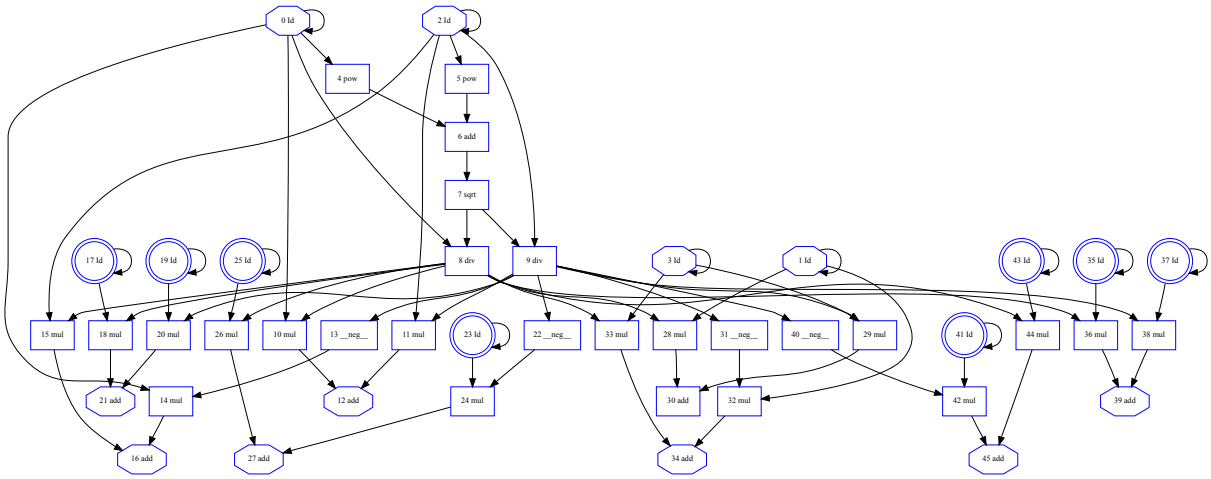



Figure 3.1.: This Figure shows the computational graph of a QR decomposition applied to a $\mathbb{R}^{2\times 2}$ matrix.

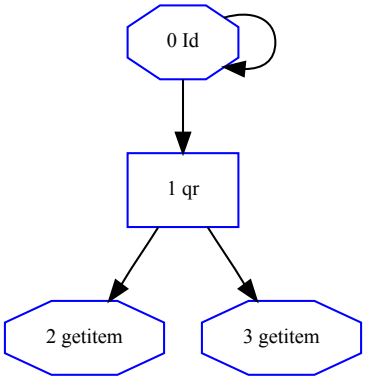



Figure 3.2.: Computational graph when the QR algorithm is considered to be a black box. The getitem nodes extract the elements $Q$ and $R$ from the tuple $(Q, R)$.

algorithms for the Schur and singular value decomposition that are based on the analytical derivatives.

In conclusion that means that a higher-level approach

- is arguably easier to implement once algorithms have been derived
- can reuse existing libraries
- can be more efficient due to highly optimized implementations
- requires less memory in the reverse mode in a natural way
- provides a framework also for functions where the algorithm does not include all necessary information.

## 3.3. Preliminaries and General Approach

The fundamental observation for the remainder of this chapter is that matrix decompositions can be written as solutions of nonlinear systems of equations

$$0 = F(x, y) \ ,$$

where $x \in \mathbb{R}^N$ is a vector containing the entries of the input matrices and $y \in \mathbb{R}^M$ a vector comprising the entries of the decomposition's output matrices. Under some regularity assumptions it is possible to use the implicit function theorem (Proposition 2.2.8) to make statements regarding the differentiability of matrix elements. When a decomposition is not unique such a statement is not directly possible since the mapping $x \mapsto y$ is not a function. On the other hand, an algorithm is a deterministic procedure. Thus, even if the mathematical mapping is non-unique and therefore not well-defined, the mapping realized by the algorithm can be well-defined. Granted the algorithm only contains differentiable statements it also follows that the algorithmic realization of the function is differentiable by virtue of the chain rule.

**Definition 3.3.1** (Identity Matrix)**.** The identity matrix is denoted

$$\mathbf{I} \equiv \mathbf{I}_N := \begin{pmatrix} 1 & & \\ & \ddots & \\ & & 1 \end{pmatrix} \in \mathbb{R}^{N \times N} .$$

**Definition 3.3.2** (block diagonal matrix)**.** The square matrix $A \in \mathbb{R}^{N \times N}$ is said to be a *block diagonal matrix* with the blocks defined by $b \in \mathbb{N}^{N_\mathrm{b}+1}$ if only the submatrices $A_{\mathrm{sl,sl}}$ have non-zero entries, where $\mathrm{sl} = b_{n_\mathrm{b}} : b_{n_\mathrm{b}+1} - 1$ is a slice index as defined in Definition 2.2.3. $N_\mathrm{b}$ is the number of blocks in the matrix.

**Example 3.3.1.** The matrix

$$A = \begin{pmatrix} * & * & * & & \\ * & * & * & & \\ * & * & * & & \\ & & & o & o \\ & & & o & o \end{pmatrix}$$

has $N_\mathrm{b} = 2$ and $b = [1, 4, 6]$. Furthermore,

$$A_{\mathrm{sl,sl}} = \begin{pmatrix} * & * & * \\ * & * & * \\ * & * & * \end{pmatrix}$$

for $\mathrm{sl} = 1 : 3$.

Matrix factorizations typically define some sparsity structure. E.g., in the QR decomposition the matrix $R$ is upper triangular. The following set of projectors are useful to enforce that matrices are diagonal or strictly lower/upper triangular matrices.

**Definition 3.3.3** (skeletal projectors)**.** Let $P_D, P_L, P_U, P_b \in \mathbb{R}^{N \times N}$. The *diagonal projector* $P_D$ is defined to be the matrix with elements $(P_D)_{ij} = \delta_{ij}$. The *strictly lower triangular projector* $P_L$ is defined to be the matrix with elements $(P_L)_{ij} = \delta_{i>j}$. Correspondingly, the matrix elements of the *strictly upper triangular projector* $P_U$ satisfy $(P_U)_{ij} = \delta_{i<j}$. The projector $P_b$ is a block diagonal matrix where $b \in \mathbb{R}^{N_\mathrm{b}+1}$ defines the block sizes. Explicitly, $(P_b)_{ij} = \sum_{n_\mathrm{b} \in N_\mathrm{b}+1} \delta_{b_{n_\mathrm{b}} \leq i < b_{n_\mathrm{b}+1}} \delta_{b_{n_\mathrm{b}} \leq j < b_{n_\mathrm{b}+1}}$. These projectors are called *skeletal projectors* to indicate that they project onto a certain skeletal structure.

**Example 3.3.2** (skeletal projectors)**.** Let $N = 3$ and $b = (1, 3, 4)$, then the skeletal projectors $P_D, P_L, P_U$ and $P_b$ are

$$P_D = \begin{pmatrix} 1 & & \\ & 1 & \\ & & 1 \end{pmatrix} \quad P_L = \begin{pmatrix} 0 & & \\ 1 & 0 & \\ 1 & 1 & 0 \end{pmatrix} \quad P_U = \begin{pmatrix} 0 & 1 & 1 \\ & 0 & 1 \\ & & 0 \end{pmatrix} \quad P_b = \begin{pmatrix} 1 & 1 & \\ 1 & 1 & \\ & & 1 \end{pmatrix} .$$

**Definition 3.3.4** (transpose, symmetric and orthogonal matrices )**.** The *transpose* of $X \in \mathbb{R}^{M\times N}$ is defined as the matrix $Y \in \mathbb{R}^{N\times M}$ s.t. $X_{mn} = Y_{nm}$ for all $m = 1,\dots,M$, $n = 1,\dots,N$. The functional dependence is denoted

$$Y = \text{transpose}(X) \equiv X^T$$

The matrix $X \in \mathbb{R}^{N\times N}$ is called *symmetric* if and only if

$$X = X^T$$

and called *orthogonal* if and only if

$$X^T X = \mathbf{1}\,.$$

### 3.3.1. Univariate Taylor Polynomial Arithmetic

One would like to know how a smooth input path $x(t) \in \mathbb{R}^N$ propagates through a matrix decomposition. Since the matrix decomposition is defined by an algebraic equation $0 = F(x, y)$ one therefore asks for a path $y(t)$ that satisfies

$$0 = F(x(t), y(t))\,.$$

I.e., the elements of the inputs $x(t)$ and outputs $y(t)$ are identified with elements of the input and output matrices.

It is necessary to generalize the notation for the following discussion. A univariate Taylor polynomial with matrix coefficients is written as

$$[X]_D = [X_{[0]},\dots,X_{[D-1]}] = \sum_{d=0}^{D-1} X_{[d]}T^d\,, \tag{3.5}$$

where $X_{[d]} \in \mathbb{R}^{M\times N}$. The set of such polynomial matrices is denoted

$$\mathbb{R}[T]/(T^D)^{M\times N} := \{\sum_{d=0}^{D-1} X_{[d]}T^d : X_{[d]} \in \mathbb{R}^{M\times N}\}\,. \tag{3.6}$$

By

$$[X_{mn}]_D := \sum_{d=0}^{D-1} X_{[d];mn}T^d \in \mathbb{R}[T]/(T^D)$$

it is meant that the element from the $m$-th row and $n$-th column is accessed. The basic observation is that a polynomial with matrix coefficients is equivalent to a matrix with polynomials as elements:

$$\sum_{d=0}^{D-1}\begin{pmatrix} X_{[d];11} & \dots & \dots X_{[d];1N} \\ \vdots & \ddots & \vdots \\ X_{[d];M1} & \dots & X_{[d];MN}\end{pmatrix}T^d = \begin{pmatrix} \sum_{d=0}^{D-1} X_{[d];11}T^d & \dots & \sum_{d=0}^{D-1} X_{[d];1N}T^d \\ \vdots & \ddots & \vdots \\ \sum_{d=0}^{D-1} X_{[d];M1}T^d & \dots & \sum_{d=0}^{D-1} X_{[d];MN}T^d\end{pmatrix}$$

To distinguish both cases the first option is called *Univariate Taylor Polynomials over Matrices* (UTPM) whereas the other case is called *Univariate Taylor Polynomials over Scalars* (UTPS). That means one can find out how coefficients $Y_{[d],ij}$ depend on $X_{[d];kl}$ using matrix valued polynomials.

It is convenient to generalize the notion of symmetric, orthogonal matrices as defined in Definition 3.3.4 etc. to UTPMs.

**Definition 3.3.5** (transpose, symmetric, orthogonal and structured UTPMs)**.** Let $[X]_D \in$

$\mathbb{R}/(T^D)^{N\times N}$ be given. The *transpose* of $[X]_D$ is defined as the matrix $[Y]_D \in \mathbb{R}/(T^D)^{N\times N}$ s.t. $X_{[d];mn} = Y_{[d];nm}$ for all $d = 0,\ldots,D-1$,$m = 1,\ldots,M$, $n = 1,\ldots,N$ and denoted

$$[X]_D^T := [Y]_D$$

$[X]_D$ is called *symmetric* if

$$[X]_D = [X]_D^T$$

and *orthogonal* if

$$[X]_D^T[X]_D \overset{D}{=} \mathbf{I}\,.$$

One says $[X]$ is upper (lower,strictly upper, strictly lower) triangular if all coefficients $X_{[d]}$ are upper (lower, strictly upper, strictly lower) triangular.

### 3.3.2. Reverse Mode

Just as the inherent structure of matrix operations can be exploited for univariate Taylor polynomial arithmetic, it is also possible to describe pullbacks of linear forms in terms of matrix operations. To gain some insight what this means, consider the function $y = \mathrm{tr}(X^{-1})$, where $X \in \mathbb{R}^{N\times N}$. It can be expressed in three-part form

| | | | | | |
|---|---|---|---|---|---|
| independent | $v_0$ | $=$ | $X$ | | |
| | $v_1$ | $=$ | $\phi_1(v_0)$ | $=$ | $\mathrm{inv}(v_0)$ |
| | $v_2$ | $=$ | $\phi_2(v_1)$ | $=$ | $\mathrm{tr}(v_1)$ |
| dependent | $y$ | $=$ | $v_2$ | | |

.

Just as described in Chapter 2.5 one would like, for given $\bar{y}$, to find $\bar{X} \in \mathbb{R}^{N\times N}$ such that

$$\bar{y}\dot{y} = \sum_{i=1}^{N}\sum_{j=1}^{N} \bar{X}_{ij}\dot{X}_{ij} = \mathrm{tr}(\bar{X}^T\dot{X}) \tag{3.7}$$

is satisfied. Successive pullbacks of $\bar{y}\dot{y}$ to the three-part form can be done by hand:

$$\bar{v}_2\dot{v}_2 = \bar{v}_2\,\mathrm{tr}(\dot{v}_1) = \mathrm{tr}(\underbrace{\mathbf{I}\bar{v}_2}_{=:\bar{v}_1^T\in\mathbb{R}^{N\times N}}\dot{v}_1) = \mathrm{tr}(\underbrace{-v_1\bar{v}_1^Tv_1}_{=:\bar{v}_0^T}\dot{v}_0)$$

and thus $\bar{X} = -v_1^T\bar{v}_1v_1^T$. One should note that the bar values $\bar{v}_l$, $l = 0,1,2$ are numerical values and not expressions. On the other hand, in this example one could exploit some additional structure if the bar values are at first collected in symbolic form. The reason is that $\bar{v}_1$ is a multiple of the identity matrix and can therefore commutes with other matrices. Thus it is possible to save one matrix multiplication and finds that $\bar{X} = -\bar{v}_2v_1^Tv_1^T$.

## 3.4. Solving Linear Systems

Let $A \in \mathbb{R}^{N\times N}$ and $B \in \mathbb{R}^{N\times M}$ be given and one would like to know the matrix $X \in \mathbb{R}^{N\times M}$ that solves $AX = B$. The defining equation of this system is

$$0 = AX - B\,. \tag{3.8}$$

and its functional dependence is denoted

$$X = \mathrm{solve}(A,B)\,. \tag{3.9}$$

### 3.4.1. Pushforward

In univariate Taylor polynomial arithmetic one needs to find $[X]_D$ such that

$$0 \overset{D}{=} [F]_D = [A]_D[X]_D - [B]_D \tag{3.10}$$

is satisfied.

**Proposition 3.4.1.** *Let* $1 \leq E \leq D$, $[A]_{D+E} \in \mathbb{R}[T]/(T^{D+E})^{N\times N}$, $[B]_{D+E} \in \mathbb{R}[T]/(T^{D+E})^{N\times M}$ *and* $[X]_D \in \mathbb{R}[T]/(T^D)^{N\times M}$ *be given s.t.* $0 \overset{D}{=} [A]_D[X]_D - [B]_D$ *holds. Denote the solution* $[\Delta X]_E$ *of the system*

$$[A]_E[\Delta X]_E \overset{E}{=} [\Delta B]_E - [\Delta F]_E - [\Delta A]_E[X]_E \,, \tag{3.11}$$

*where*

$$[\Delta F]_E T^D \overset{D+E}{:=} [A]_D[X]_D - [B]_D \,.$$

*Then* $[X]_{D+E} := [X]_D + [\Delta X]_E T^D$ *satisfies*

$$0 \overset{D+E}{=} [A]_{D+E}[X]_{D+E} - [B]_{D+E} \,.$$

*Proof.* The following three equations are equivalent:

$$\begin{aligned} &\Big([A]_D + [\Delta A]_E T^D)\Big)\Big([X]_D + [\Delta X]_E T^D)\Big) \overset{D+E}{=} [B]_D + [\Delta B]_E T^D \\ \Leftrightarrow\ & [\Delta F]_E T^D + \left([\Delta A]_E[X]_D + [A]_D[\Delta X]_E\right) T^D \overset{D+E}{=} [\Delta B]_E T^D \\ \Leftrightarrow\ & [A]_E[\Delta X]_E \overset{E}{=} [\Delta B]_E - [\Delta F]_E - [\Delta A]_E[X]_E \,. \end{aligned}$$

This already concludes the proof. □

*Remark.* Note that the polynomial operations of the polynomial factor ring $\mathbb{R}[T]/(T^D)$ are lifted to $\mathbb{R}[T]$ and then reduced modulo $\mathbb{R}[T]/(T^{D+E})$ (see Defintion 2.3.7). Consider for instance the computation of $[\Delta F]_E T^D \overset{D+E}{:=} [A]_D[X]_D - [B]_D$. The operation $[A]_D[X]_D$ is evaluated in $\mathbb{R}[T]/(T^{D+E})$ and then, indicated by $\overset{D+E}{=}$, reduced modulo $\mathbb{R}[T]/(T^{D+E})$.

This proposition can now be used within an algorithm to compute all coefficients. For the special case $E = 1$ is is shown in Algorithm 3.4.1.

**input** : $[A]_D = [A_{[0]}, \ldots, A_{[D-1]}]$, where $A_{[d]} \in \mathbb{R}^{N\times N}$, $d = 0, \ldots, D-1$
**input** : $[B]_D = [B_{[0]}, \ldots, B_{[D-1]}]$, where $B_{[d]} \in \mathbb{R}^{N\times M}$, $d = 0, \ldots, D-1$
**output**: $[X]_D = [X_{[0]}, \ldots, X_{[D-1]}]$, where $X_{[d]} \in \mathbb{R}^{N\times M}$, $d = 0, \ldots, D-1$

$X_{[0]} = \text{solve}(A_{[0]}, B_{[0]})$

**for** $d = 1$ **to** $D-1$ **do**
  $X_{[d]} = \text{solve}(A_{[0]}, B_{[d]} - \sum_{k=1}^{d} A_{[k]} X_{[d-k]})$
**end**

**Algorithm 3.4.1:** This algorithm solves the linear system $[A]_D[X]_D \overset{D}{=} [B]_D$ by sequential Hensel lifting. It corresponds to the case $E = 1$ in Propostion 3.4.1.

### 3.4.2. Pullback

In Chapter 3.3.2 it was discussed that NLA functions can be regarded as atomic instructions. If one would like to apply the reverse mode of AD, as explained in Chapter 2.5.1, one has

ultimately to find an $\bar{A}$ and a $\bar{B}$ such that

$$\mathrm{tr}(\bar{X}^T \dot{X}) = \mathrm{tr}(\bar{A}^T \dot{A}) + \mathrm{tr}(\bar{B}^T \dot{B}) \,,$$

holds when $\dot{X} = \partial \mathrm{solve}(A, B)\{\dot{A}, \dot{B}\}$. $\bar{X}$ is a given quantity. If one applies matrix calculus to the defining equation one obtains

$$\begin{aligned} \dot{B} =& \dot{A}X + A\dot{X} \Leftrightarrow A^{-1}(\dot{B} - \dot{A}X) = \dot{X} \\ \mathrm{tr}(\bar{X}^T \dot{X}) =& \mathrm{tr}(\bar{X}^T A^{-1} \dot{B}) + \mathrm{tr}(-\bar{X}^T A^{-1} \dot{A} X) = \mathrm{tr}(\bar{T}^T \dot{B}) + \mathrm{tr}(\bar{A}^T \dot{A}) \,, \end{aligned}$$

where $A^T \bar{T} = \bar{X}$ and $-X\bar{X}^T A^{-1} = -X\bar{T}^T$. Therefore

$$\bar{B} = \bar{T} \quad \text{and} \quad \bar{A} = -\bar{T}X^T \,.$$

One should note that in a larger computational context it is possible that $A$ or $B$ could be input to more than one function. In that case it is necessary to increment the current value of $\bar{A}$ and $\bar{B}$ as follows:

$$\bar{B} = \bar{B} + \bar{T} \quad \text{and} \quad \bar{A} = \bar{A} - \bar{T}X^T \,.$$

See Chapter 2.5.1 for a more detailed explanation.

As described in Chapter 2.5.2 it is possible to evaluate the pullback formulas in UTP arithmetic and obtains Algorithm 3.4.2.

**input** : $[A]_D = [A_{[0]}, \ldots, A_{[D-1]}]$, where $A_{[d]} \in \mathbb{R}^{N \times N}$, $d = 0, \ldots, D-1$
**input** : $[B]_D = [B_{[0]}, \ldots, B_{[D-1]}]$, where $B_{[d]} \in \mathbb{R}^{N \times M}$, $d = 0, \ldots, D-1$
**input** : $[X]_D = [X_{[0]}, \ldots, X_{[D-1]}]$, where $X_{[d]} \in \mathbb{R}^{N \times M}$, $d = 0, \ldots, D-1$
**input/output**: $[\bar{A}]_D = [\bar{A}_{[0]}, \ldots, \bar{A}_{[D-1]}]$, where $\bar{A}_{[d]} \in \mathbb{R}^{N \times N}$, $d = 0, \ldots, D-1$
**input/output**: $[\bar{B}]_D = [\bar{B}_{[0]}, \ldots, \bar{B}_{[D-1]}]$, where $\bar{B}_{[d]} \in \mathbb{R}^{N \times M}$, $d = 0, \ldots, D-1$
**input** : $[\bar{X}]_D = [\bar{X}_{[0]}, \ldots, \bar{X}_{[D-1]}]$, where $\bar{X}_{[d]} \in \mathbb{R}^{N \times M}$, $d = 0, \ldots, D-1$

$[\bar{T}]_D = \mathrm{solve}([A]_D^T, [\bar{X}]_D)$
$[\bar{A}]_D = [\bar{A}]_D - [\bar{T}]_D [X]_D^T$
$[\bar{B}]_D = [\bar{B}]_D + [\bar{T}]_D$

**Algorithm 3.4.2:** Reverse accumulation in univariate Taylor polynomial arithmetic.

## 3.5. Matrix Inversion

Very similar to the solution of linear systems is the matrix inversion. It corresponds to the case $B = \mathbf{I}$. The *defining equation of the matrix inversion* $Y = X^{-1}$ is

$$0 = XY - \mathbf{I} \,, \tag{3.12}$$

and the functional dependence is written as

$$Y = \mathrm{inv}(X) \equiv X^{-1} \,. \tag{3.13}$$

### 3.5.1. Pushforward

**Proposition 3.5.1.** *Let* $[X]_{D+E}$ *and* $[Y]_D$ *be given such that* $0 \overset{D}{=} [Y]_D[X]_D - \mathbf{1}$ *is satisfied. Then* $[Y]_{D+E} = [Y]_D + [\Delta Y]_E T^D$ *satisfies* $0 \overset{D+E}{=} [Y]_{D+E}[X]_{D+E} - \mathbf{1}$ *if*

$$[\Delta F]_E T^D \overset{D+E}{=} [X]_D[Y]_D - \mathbf{1} \tag{3.14}$$

$$[\Delta Y]_E \overset{E}{=} -[Y]_E([\Delta F]_E + [\Delta X]_E[Y]_E) \,. \tag{3.15}$$

*Proof.* A straight-forward calculation starting at the defining equation yields

$$\begin{aligned} 0 &\overset{D+E}{=} ([X]_D + [\Delta X]_E T^D)([Y]_D + [\Delta Y]_E T^D) - \mathbf{1} \\ &\overset{D+E}{=} [\Delta F]_E T^D + ([X]_D[\Delta Y]_E + [\Delta X]_E[Y]_D)T^D \\ &\overset{E}{=} [\Delta F]_E + [X]_D[\Delta Y]_E + [\Delta X]_E[Y]_D \\ \therefore \quad [\Delta Y]_E &= -[Y]_E([\Delta F]_E + [\Delta X]_E[Y]_E) \,. \end{aligned}$$

□

**input** : $[X]_D = [X_{[0]}, \ldots, X_{[D-1]}]$, where $X_{[d]} \in \mathbb{R}^{N \times N}$, $d = 0, \ldots, D-1$
**output**: $[Y]_D = [Y_{[0]}, \ldots, Y_{[D-1]}]$, where $Y_{[d]} \in \mathbb{R}^{N \times N}$, $d = 0, \ldots, D-1$
$Y_{[0]} = \mathrm{inv}(X_{[0]})$
**for** $d = 1$ **to** $D-1$ **do**
  $Y_{[d]} = -Y_{[0]} \left( \sum_{k=0}^{d-1} X_{[d-k]} Y_{[k]} \right)$
**end**

**Algorithm 3.5.1:** This algorithm inverts a polynomial matrix $[Y]_D \overset{D}{=} [X]_D^{-1}$ by sequential Hensel lifting. It corresponds to the case $E = 1$ in Propostion 3.5.1.

### 3.5.2. Pullback

The pullback of the linear form is also based on the defining equations: $Y = X^{-1}$, $0 = XY - \mathbf{1}$ and hence $0 = \dot{X}Y + X\dot{Y}$, $\dot{Y} = -X^{-1}\dot{X}Y = -Y\dot{X}Y$. Thus $\mathrm{tr}(\bar{Y}^T\dot{Y}) = \mathrm{tr}(-\bar{Y}^T Y\dot{X}Y) = \mathrm{tr}(-Y\bar{Y}^T Y\dot{X})$ and therefore

$$\bar{X} = -Y^T\bar{Y}Y^T \,. \tag{3.16}$$

One can evaluate this update rule in univariate Taylor polynomial arithmetic, see Algorithm 3.5.2.

**input** : $[X]_D = [X_{[0]}, \ldots, X_{[D-1]}]$, where $X_{[d]} \in \mathbb{R}^{N \times N}$, $d = 0, \ldots, D-1$
**input** : $[Y]_D = [Y_{[0]}, \ldots, Y_{[D-1]}]$, where $Y_{[d]} \in \mathbb{R}^{N \times N}$, $d = 0, \ldots, D-1$
**output**: $[\bar{X}]_D = [\bar{X}_{[0]}, \ldots, \bar{X}_{[D-1]}]$, where $\bar{X}_{[d]} \in \mathbb{R}^{N \times N}$, $d = 0, \ldots, D-1$
**input** : $[\bar{Y}]_D = [\bar{Y}_{[0]}, \ldots, \bar{Y}_{[D-1]}]$, where $\bar{Y}_{[d]} \in \mathbb{R}^{N \times N}$, $d = 0, \ldots, D-1$
$[\bar{X}]_D = [\bar{X}]_D - [Y]_D^T[\bar{Y}]_D[Y]_D^T$

**Algorithm 3.5.2:** This algorithm performs a pullback of the matrix inversion in univariate Taylor polynomial arithmetic.

## 3.6. Cholesky Decomposition

The Cholesky decomposition can be used to solve the linear system $Ax = b$ or to compute $\det(A)$ when $A$ is a symmetric positive definite matrix. Smith [1995] describes how AD techniques on the level of basic elementary functions can be applied to the Cholesky algorithm itself and derives algorithms useful for the forward and reverse mode. Here, algorithms based on the defining equations of the Cholesky decomposition are derived.

**Definition 3.6.1** (Cholesky Decomposition)**.** The *Cholesky Decomposition* is the solution of the implicit system

$$0 = LL^T - A \tag{3.17}$$
$$0 = P_R \circ L \ , \tag{3.18}$$

where $L \in \mathbb{R}^{N\times N}$ lower triangular matrix, $A \in \mathbb{R}^{N\times N}$ symmetric positive definite matrix and $(P_R)_{ij} := \delta_{i<j}$ the projector to the strictly upper right matrices, i.e., the element-wise multiplication $\circ$ with $P_R$ selects all entries above the diagonal.

**Proposition 3.6.1** (existence and uniqueness)**.** *The Cholesky decomposition of a real symmetric positive definite matrix $A \in \mathbb{R}^{N\times N}$ exists and is unique.*

*Proof.* See the textbook by Datta [1998]. □

### 3.6.1. Pushforward

**Proposition 3.6.2.** *Let $[A]_{D+E} \in \mathbb{R}[T]/(T^{D+E})^{N\times N}$ with symmetric positive definite $A_{[0]}$ and symmetric $A_{[d]}$ ($d \geq 1$) as well as $[L]_D \in \mathbb{R}[T]/(T^D)^{N\times N}$ with $L_{[d]}$ lower triangular for $d \geq 0$ be given. Then a closed-form solution for $[\Delta L]_E$ is given by*

$$\begin{aligned}
[\Delta\tilde{F}]_E T^D &\overset{D+E}{:=} [L]_D [L]_D^T - [A]_D \\
[\Delta F] &\overset{E}{:=} [L]_E^{-1}([\Delta\tilde{F}]_E - [\Delta A]_E)[L]_E^{-T} \\
[\Delta L]_E &\overset{E}{=} -[L]_E \left( (P_L + \frac{1}{2}P_D) \circ [\Delta F]_E \right) \ ,
\end{aligned}$$

*Proof.* For notational simplicity the formal identification $A \equiv [A]_D$ and $\Delta A \equiv [A]_{D:D+E}$, $\Delta L \equiv [L]_{D:D+E}$ is used. Depending on the context $L \equiv [L]_D$ or $L \equiv [L]_E$. When $L$ appears in a equation modulo $(T^E)$ the higher-order coefficients are neglected and can therefore be discarded. Applying Hensel's lifting lemma to (3.17) yields

$$\begin{aligned}
0 &\overset{D+E}{=} (L + \Delta L T^D)(L^T + \Delta L^T T^D) - (A + \Delta A T^D) \\
&\overset{D+E}{=} (LL^T - A) + (L\Delta L^T + \Delta L L^T - \Delta A)T^D \\
&\overset{E}{=} \Delta\tilde{F} - \Delta A + L\Delta L^T + \Delta L L^T \ ,
\end{aligned}$$

where $\Delta\tilde{F}T^D \overset{D+E}{:=} (LL^T - A)$ has been defined. Since $L$ is nonsingular one can multiply $L^{-1}$ from the left and $L^{-T}$ from the right. That $L$ is invertible is guaranteed by $A_{[0]}$ symmetric positive definite. One thus obtains

$$\begin{aligned}
0 &\overset{E}{=} L^{-1}(\Delta\tilde{F} - \Delta A)L^{-T} + \Delta L^T L^{-T} + L^{-1}\Delta L \\
&\overset{E}{=} \Delta F + L^{-1}\Delta L + (L^{-1}\Delta L)^T \ ,
\end{aligned} \tag{3.19}$$

where $\Delta F \overset{E}{:=} L^{-1}(\Delta\tilde{F} - \Delta A)L^{-T}$. It is possible to solve for the diagonal elements of $\Delta L$, i.e.

$$\begin{aligned} 0 &\overset{E}{=} P_D \circ \left(\Delta F + L^{-1}\Delta L + (L^{-1}\Delta L)^T\right) \\ &\overset{E}{=} P_D \circ (\Delta F) + 2P_D \circ (L^{-1}\Delta L) \\ &\overset{E}{=} P_D \circ (\Delta F) + 2P_D \circ (L^{-1})P_D \circ (\Delta L) \\ P_D \circ (\Delta L) &\overset{E}{=} -\frac{1}{2}(P_D \circ L)(P_D \circ (\Delta F)) \,, \end{aligned} \tag{3.20}$$

where the Lemmas B.0.13, B.0.14 and B.0.15 have been used. Similarly, the off-diagonal entries of $\Delta L$ are obtained:

$$\begin{aligned} 0 &\overset{E}{=} (P_L + P_D) \circ \left(\Delta F + L^{-1}\Delta L + (L^{-1}\Delta L)^T\right) \\ &\overset{E}{=} (P_L + P_D) \circ (\Delta F) + L^{-1}\Delta L + P_D \circ (L^{-1}\Delta L)^T \\ &\overset{E}{=} (P_L + P_D) \circ (\Delta F) + L^{-1}\Delta L + (P_D \circ L)^{-1}(P_D \circ \Delta L) \\ &\overset{E}{=} (P_L + P_D) \circ (\Delta F) + L^{-1}(P_D \circ \Delta L) + L^{-1}(P_L \circ \Delta L) + (P_D \circ L)^{-1}(P_D \circ \Delta L) \\ P_L \circ (\Delta L) &\overset{E}{=} -L\left((P_L + P_D) \circ (\Delta F) + (P_D \circ L)^{-1}(P_D \circ \Delta L)\right) - P_D \circ \Delta L \,. \end{aligned}$$

Here, (3.18) and as well as Lemmas B.0.13 B.0.14, B.0.15 have been applied. Finally, the last expression can be simplified by using (3.20) to

$$\begin{aligned} P_L \circ (\Delta L) &\overset{E}{=} -L\left(P_L \circ \Delta F + \frac{1}{2}P_D \circ \Delta F\right) - P_D \circ \Delta L \\ &\overset{E}{=} -L\left((P_L + \frac{1}{2}P_D) \circ \Delta F\right) - P_D \circ \Delta L \,. \end{aligned} \tag{3.21}$$

Thus, adding $P_D \circ \Delta L$ to both sides in the last equation results in

$$[\Delta L]_E \overset{E}{=} -[L]_E\left((P_L + \frac{1}{2}P_D) \circ [\Delta F]_E\right) \,.$$

□

**input** : $[A]_D = [A_{[0]}, \ldots, A_{[D-1]}]$, where $A_{[d]} \in \mathbb{R}^{N\times N}$ symmetric positive definite, $d = 0, \ldots, D-1$
**output**: $[L]_D = [L_{[0]}, \ldots, L_{[D-1]}]$, where $L_{[d]} \in \mathbb{R}^{N\times N}$ lower triangular, $d = 0, \ldots, D-1$

$L_{[0]} = \text{cholesky}(A_{[0]})$

**for** $d = 1$ **to** $D-1$ **do**
  $\Delta F = L_{[0]}^{-1}(\sum_{k=1}^{d-1} L_{[d-k]}L_{[k]}^T - A_{[d]})L_{[0]}^{-T}$
  $L_{[d]} = -L_{[0]}\left((P_L + \frac{1}{2}P_D) \circ \Delta F\right)$
**end**

**Algorithm 3.6.1:** This algorithm performs a Cholesky decomposition in univariate Taylor polynomial arithmetic. I.e., this algorithm considers the case $E = 1$ in the Proposition 3.6.2.

### 3.6.2. Pullback

For the reverse mode of AD it is necessary to pullback the form $\mathrm{tr}(\bar{L}^T \dot{L})$ where $\dot{L} = \partial\,\mathrm{cholesky}(A)\{\dot{A}\}$.

**Proposition 3.6.3.** *Given symmetric positive definite $A \in \mathbb{R}^{N\times N}$ with $L = \mathrm{cholesky}(A)$ and a lower triangular matrix $\bar{L} \in \mathbb{R}^{N\times N}$. Furthermore, let $\dot{L} = \partial\,\mathrm{cholesky}(A)\{\dot{A}\}$. Then $\mathrm{tr}(\bar{L}^T \dot{L}) = \mathrm{tr}(\bar{A}^T \dot{A})$ is satisfied by*

$$\bar{A} = \frac{1}{2} L^{-T} \left(B + B^T\right) L^{-1} , \tag{3.22}$$

$$\text{where} \quad B = (P_L + \frac{1}{2} P_D) \circ (L^T \bar{L}) . \tag{3.23}$$

*Proof.* The proof is similar to the proof of the pushforward. However, there is an additional structure that has to be satisfied: $A$ is symmetric and therefore $\dot{A}$ and $\bar{A}$ are also symmetric. Differentiation of the implicit system $0 = LL^T - A$ and multiplying by $L^{-1}$ ($L^{-T}$) from the left (right) yields

$$0 = L^{-1}\dot{L} + \dot{L}^T L^{-T} - L^{-1}\dot{A}L^{-T} .$$

The diagonal elements of $\dot{L}$ can be read from

$$\begin{aligned} 0 &= P_D \circ \left(L^{-1}\dot{L} + \dot{L}^T L^{-T} - L^{-1}\dot{A}L^{-T}\right) \\ &= 2P_D \circ (L^{-1}\dot{L}) - P_D \circ \left(L^{-1}\dot{A}L^{-T}\right) \\ P_D \circ \dot{L} &= \frac{1}{2}(P_D \circ L)(P_D \circ (L^{-1}\dot{A}L^{-T})) . \end{aligned} \tag{3.24}$$

The off-diagonal elements can be obtained from

$$\begin{aligned} 0 &= (P_L + P_D) \circ \left(L^{-1}\dot{L} + \dot{L}^T L^{-T} - L^{-1}\dot{A}L^{-T}\right) \\ &= L^{-1}\dot{L} + (P_D \circ L)^{-1}(P_D \circ \dot{L}) - (P_L + P_D) \circ (L^{-1}\dot{A}L^{-T}) \\ &= L^{-1}\dot{L} - (P_L + \frac{1}{2}P_D) \circ (L^{-1}\dot{A}L^{-T}) \\ \dot{L} &= L(P_L + \frac{1}{2}P_D) \circ (L^{-1}\dot{A}L^{-T}) , \end{aligned} \tag{3.25}$$

where 3.24 has been used in the second line of the above equation. Finally, the linear form can be transformed as follows:

$$\begin{aligned} \mathrm{tr}(\bar{L}^T \dot{L}) &= \mathrm{tr}\left(\bar{L}^T L((P_L + \frac{1}{2}P_D) \circ (L^{-1}\dot{A}L^{-T}))\right) \\ &= \mathrm{tr}\left((L^{-1}\dot{A}L^{-T})^T((P_L + \frac{1}{2}P_D) \circ (\bar{L}^T L)^T)\right) \\ &= \mathrm{tr}\left((L^{-1}\dot{A}L^{-T})^T B\right) \\ &= \frac{1}{2}\,\mathrm{tr}\left(L^{-T}BL^{-1}\dot{A}\right) + \frac{1}{2}\,\mathrm{tr}\left(L^{-T}B^T L^{-1}\dot{A}\right) \\ &= \frac{1}{2}\,\mathrm{tr}\left(L^{-T}(B + B^T)L^{-1}\dot{A}\right) \end{aligned}$$

where $\mathrm{tr}(A^T(B \circ C)) = \mathrm{tr}(C^T(B \circ A))$, $\dot{A} = \dot{A}^T$ and the cyclic invariance property of the trace have been used. This concludes the proof. □

*Remark.* The formula of $\bar{A}$ has been symmetrized in the proof of Proposition 3.6.3. The fact that $A$ is symmetric, i.e., $A_{ij} = A_{ji}$ for all $i, j = 1, \ldots, N$ means that also $\dot{A}_{ij} = \dot{A}_{ji}$. Thus, in the computation of the trace one has the simple identity $\mathrm{tr}(\bar{A}^T \dot{A}) = \sum_{i,j=1}^{N} \bar{A}_{ij}\dot{A}_{ij} = \sum_{1 \le j \le i \le N}(1 - \frac{1}{2}\delta_{ij})(\bar{A}_{ij} + \bar{A}_{ji})\dot{A}_{ij}$ when $A$ is defined by its lower triangular part. Also, when $A = A(x)$ then $\mathrm{tr}(\bar{A}\dot{A}(x)) = \sum_{ij} \bar{A}_{ij}\dot{A}_{ij} = \sum_{ij} \bar{A}_{ij} \sum_k \frac{\dot{A}_{ij}}{\dot{x}_k}\dot{x}_k = \sum_k (\sum_{ij} \bar{A}_{ij} \frac{\dot{A}_{ij}}{\dot{x}_k})\dot{x}_k$. Since $\frac{\dot{A}_{ij}}{\dot{x}_k} = \frac{\dot{A}_{ji}}{\dot{x}_k}$ the result is the same if $\bar{A}$ is symmetrized or not.

## 3.7. Full QR Decomposition

A QR decomposition is a factorization of the form $A = QR$, where $A \in \mathbb{R}^{M\times N}$. When $M \neq N$, i.e., when $A$ is a tall matrix, there are two possibilities: Either $Q \in \mathbb{R}^{M\times M}$ and $R \in \mathbb{R}^{M\times N}$ or $Q \in \mathbb{R}^{M\times N}$ and $R \in \mathbb{R}^{N\times N}$. In the following, the first option is called the *full QR decomposition* and the second the *thin QR decomposition*. The full QR decomposition can be used to find the nullspace of a matrix $A \in \mathbb{R}^{M\times N}$ where $M \leq N$ by computing $(Q_1, Q_2)R = (A^T, 0_{N,N-M})$. Then $Q_2$ spans the nullspace of $A$, i.e., $0 = AQ_2$.

**Definition 3.7.1** (full QR decomposition)**.** Let $A \in \mathbb{R}^{M\times N}$, then the factorization $A = QR$ is called the *full* QR decomposition of $A$ if

$$\begin{aligned} 0 =& QR - A \\ 0 =& Q^TQ - \mathbf{1}_M \\ 0 =& P_L \circ R \end{aligned}$$

In other words, the columns of $Q \in \mathbb{R}^{M\times M}$ are mutually orthonormal and the matrix $R \in \mathbb{R}^{M\times N}$ is upper triangular, i.e., $R_{:N,:}$ is upper triangular and $R_{N+1:,:} = 0$. The functional dependence is denoted

$$Q, R = \text{qrfull}(A) \ . \tag{3.26}$$

### 3.7.1. Pushforward

To compute $[Q]_D, [R]_D = E_D(\text{qrfull})([A]_D)$ one can apply Newton-Hensel lifting (2.14) to solve

$$0 \overset{D}{=} [Q]_D[R]_D - [A]_D \tag{3.27}$$

$$0 \overset{D}{=} [Q]_D^T[Q]_D - \mathbf{1} \tag{3.28}$$

$$0 \overset{D}{=} P_L \circ [R]_D \ . \tag{3.29}$$

However, one should avoid the explicit construction of the Jacobian $F_y$ since it is a sparse matrix with additional structure. Rather, one assumes that one has already computed $[Q]_D$ and $[R]_D$ and computes the next $1 \leq E \leq D$ coefficients by performing a first order Taylor expansion $[Q]_{D+E} = [Q]_D + [\Delta Q]_E T^D$ and $[R]_{D+E} = [R]_D + [\Delta R]_E T^D$ and tries to solve for the yet unknown $[\Delta R]_E$ and $[\Delta Q]_E$. As result one obtains Proposition 3.7.1.

**Proposition 3.7.1.** *Let $[A]_{D+E} \in \mathbb{R}[T]/(T^{D+E})^{M\times N}$ with $M \geq N$ and $1 \leq E \leq D$, $[R]_D \in \mathbb{R}[T]/(T^D)^{M\times N}$ where $[R_{:N,:}]_D$ is upper triangular with nonsingular $R_{[0],:N,:}$ and $[Q]_D \in \mathbb{R}^{M\times M}[T]/(T^D)$ orthogonal be given and satisfy the defining equations of order $D$. Then $[\Delta R_{:N,:}]_E \equiv [R_{:N,:}]_{D:D+E-1}$ and $[\Delta Q]_E \equiv [Q]_{D:D+E-1}$ are given by*

$$\begin{aligned} [\Delta F]_E T^{D\,D+E} \overset{=}{} & - [Q]_D[R]_D + [A]_{D+E} \\ [\Delta G]_E T^{D\,D+E} \overset{=}{} & - [Q]_D^T[Q]_D + \mathbf{1} \\ [S]_E \overset{E}{=} & \frac{1}{2}[\Delta G]_E \\ P_L \circ ([X_{:,:N}]_E) \overset{E}{=} & P_L \circ \left([Q]_E^T[\Delta F]_E[R_{:N,:}]_E^{-1}\right) - P_L \circ [S_{:,:N}]_E \\ [\Delta R]_E \overset{E}{=} & [Q]_E^T[\Delta F]_E - ([S]_E + [X]_E)[R]_E \\ [\Delta Q]_E \overset{E}{=} & [Q]_E\left([S]_E + [X]_E\right) \ , \end{aligned}$$

*where $P_L \in \mathbb{R}^{M\times N}$ with $(P_L)_{ij} = \delta_{j<i}$.*

*Proof.* Consider the first defining equation and try to separate the known from the unknown quantities:

$$\begin{aligned}
0 &\overset{D+E}{=} [Q]_{D+E}[R]_{D+E} - [A]_{D+E} \\
&\overset{D+E}{=} ([Q]_D + [\Delta Q]_E T^D)([R]_D + [\Delta R]_E T^D) - [A]_{D+E} \\
&\overset{D+E}{=} [Q]_D[R]_D - [A]_{D+E} + ([\Delta Q]_E[R]_D + [Q]_D[\Delta R]_E)T^D \\
&\overset{D+E}{=} -[\Delta F]_E T^D + ([\Delta Q]_E[R]_D + [Q]_D[\Delta R]_E)T^D \\
&\overset{E}{=} -[\Delta F]_E + [\Delta Q]_E[R]_E + [Q]_E[\Delta R]_E \,. \qquad (3.30)
\end{aligned}$$

Similarly for the second defining equation

$$\begin{aligned}
0 &\overset{D+E}{=} [Q]^T_{D+E}[Q]_{D+E} - \mathbf{1} \\
&\overset{D+E}{=} [Q]^T_D[Q]_D - \mathbf{1} + ([Q]^T_D[\Delta Q]_E + [\Delta Q]^T_E[Q]_D)T^D \\
\Rightarrow 0 &\overset{E}{=} -[\Delta G]_E + [Q]^T_E[\Delta Q]_E + [\Delta Q]^T_E[Q]_E \\
&\overset{E}{=} -[\Delta G]_E + [S]_E + [X]_E + [S]_E - [X]_E \\
\Rightarrow \quad [S]_E &= \frac{1}{2}[\Delta G]_E \,,
\end{aligned}$$

where $[S]_E + [X]_E = [Q]^T_E[\Delta Q]_E$ and it has been used that every matrix can be written as the sum of a symmetric and an antisymmetric matrix. Now multiply (3.30) by $[Q]^T_E$ from the left to obtain

$$\begin{aligned}
0 &\overset{E}{=} -[Q]^T_E[\Delta F]_E + [Q]^T_E[\Delta Q]_E[R]_E + [\Delta R]_E \qquad (3.31) \\
&\overset{E}{=} -[Q]^T_E[\Delta F]_E + ([S]_E + [X]_E)[R]_E + [\Delta R]_E \qquad (3.32) \\
&\overset{E}{=} -[Q]^T_E[\Delta F]_E + [S]_E[R]_E + [X]_E[R]_E + [\Delta R]_E \,.
\end{aligned}$$

Multiplication of $[R_{:N,:}]^{-1}_E$ from the right yields

$$\begin{aligned}
0 &\overset{E}{=} -[Q]^T_E[\Delta F]_E[R_{:N,:}]^{-1}_E + [S]_E[R]_E[R_{:N,:}]^{-1}_E + [X]_E[R]_E[R_{:N,:}]^{-1}_E + \\
&\quad + [\Delta R]_E[R_{:N,:}]^{-1}_E \\
&\overset{E}{=} -[Q]^T_E[\Delta F]_E[R_{:N,:}]^{-1}_E + [S_{:,:N}]_E + [X_{:,:N}]_E + [\Delta R]_E[R_{:N,:}]^{-1}_E \\
\Rightarrow P_L \circ ([X_{:,:N}]_E) &\overset{E}{=} P_L \circ \left([Q]^T_E[\Delta F]_E[R_{:N,:}]^{-1}_E - [S_{:,:N}]_E\right) \,.
\end{aligned}$$

The coefficients of $X_{:,N+1:}$ are not specified and can for instance be set to zero. Since $X$ is antisymmetric it is already defined by the above equation. From the definition $[S]_E + [X]_E \overset{E}{=} [Q]^T_E[\Delta Q]_E$ of $[S]_E$ and $[X]_E$ one can obtain $[\Delta Q]_E$ as

$$[\Delta Q]_E = [Q]_E([S]_E + [X]_E)$$

because for orthogonal $Q$ one has the identity $QQ^T = \mathbf{1}$. □

One can use Proposition 3.7.1 to derive an explicit algorithm as shown in Algorithm 3.7.1, where at each step $E = 1$ is used.

**input** : $[A]_D = [A_{[0]}, \ldots, A_{[D-1]}]$, where $A_{[d]} \in \mathbb{R}^{M \times N}$, $d = 0, \ldots, D-1$ and $\operatorname{rank}(A_{[0]}) = N$, $M \geq N$.
**output**: $[Q]_D = [Q_{[0]}, \ldots, Q_{[D-1]}]$ orthogonal, where $Q_{[d]} \in \mathbb{R}^{M \times M}$ , $d = 0, \ldots, D-1$
**output**: $[R]_D = [R_{[0]}, \ldots, R_{[D-1]}]$ upper triangular, where $R_{[d]} \in \mathbb{R}^{M \times N}$ , $d = 0, \ldots, D-1$

$Q_{[0]}, R_{[0]} = \operatorname{qrfull}(A_{[0]})$

**for** $d = 1$ **to** $D-1$ **do**
  $\Delta F = A_{[d]} - \sum_{k=1}^{d-1} Q_{[d-k]} R_{[k]}$
  $S = -\frac{1}{2} \sum_{k=1}^{d-1} Q_{[d-k]}^T Q_{[k]}$
  $X_{:,:N} = P_L \circ (Q_{[0]}^T \Delta F R_{[0];:N,:N}^{-1} - S_{:,:N})$
  $X_{:,N+1:} = 0$
  $X = X - X^T$
  $R_{[d]} = Q_{[0]}^T \Delta F - (S + X) R_{[0]}$
  $Q_{[d]} = Q_{[0]} (S + X)$
**end**

**Algorithm 3.7.1:** Sequential Hensel lifting for the QR decomposition based on Proposition 3.7.1 with $E = 1$.

### 3.7.2. Pullback

**Proposition 3.7.2** (pullback of the full QR decomposition)**.** *Let $A \in \mathbb{R}^{M \times N}$, $\bar{Q} \in \mathbb{R}^{M \times M}$ and $\bar{R} \in \mathbb{R}^{M \times N}$ upper triangular be given. Furthermore, assume $M \geq N$, $\operatorname{rank}(A) = N$, $Q, R = \operatorname{qrfull}(A)$. Then $\bar{A} \in \mathbb{R}^{M \times N}$ can be computed by*

$$\bar{A} = Q \left( \bar{R} + \left( P_L \circ \left( R\bar{R}^T - \bar{R}R^T + Q^T\bar{Q} - \bar{Q}^T Q \right) \right) R^{+T} \right) .$$

*Here, $R^+$ denotes the Moore-Penrose pseudoinverse of $R$. That means it satisfies $RR^+R = R$ and since $R$ has full column rank also $R^+R = \mathbf{1}$.*

*Proof.* Perturbation of $A$ by $\dot{A}$ in the implicit system

$$\begin{aligned} 0 &= A - QR \\ 0 &= Q^T Q - \mathbf{1} \\ 0 &= P_L \circ R \end{aligned}$$

results, to first order, in

$$\begin{aligned} 0 &= \dot{A} - \dot{Q}R - Q\dot{R} \quad (*) \\ 0 &= \dot{Q}^T Q + Q^T \dot{Q} \quad (**) . \end{aligned}$$

Define the antisymmetric matrix $X := Q^T \dot{Q}$. Multiplication of Eqn. (*) from the left with $Q^T$ yields

$$\begin{aligned} 0 &= Q^T \dot{A} - XR - \dot{R} \\ \text{hence} \quad \dot{R} &= Q^T \dot{A} - XR . \end{aligned}$$

The multipication of this last equation from the right with the Moore-Penrose pseudoinverse

$R^+ = ({R_{:N,:}}^{-1}, 0)$ yields the equivalent equation

$$\begin{aligned} 0 =& Q^T \dot{A} R^+ - X R R^+ - \dot{R} R^+ \\ \text{and thus } \; P_L \circ X =& P_L \circ (Q^T \dot{A} R^+) \; , \end{aligned}$$

where $X_{:,N+1:} = 0$ is chosen rather arbitrarily. Since $X$ is antisymmetric one obtains

$$X = (P_L \circ X) - (P_L \circ X)^T \; .$$

These results can be used to compute the pullback:

$$\begin{aligned} \mathrm{tr}(\bar{R}^T \dot{R}) + \mathrm{tr}(\bar{Q}^T \dot{Q}) &= \mathrm{tr}(Q \bar{R} \dot{A}^T) - \mathrm{tr}(R \bar{R}^T X) + \mathrm{tr}(\bar{Q}^T Q Q^T \dot{Q}) \\ &= \mathrm{tr}(Q \bar{R} \dot{A}^T) + \mathrm{tr}(\underbrace{(\bar{Q}^T Q - R \bar{R}^T)}_{=:K} X) \\ &= \mathrm{tr}(Q \bar{R} \dot{A}^T) + \mathrm{tr}((K - K^T)(P_L \circ X)) \\ &= \mathrm{tr}(Q \bar{R} \dot{A}^T) + \mathrm{tr}(R^{+T} \dot{A}^T Q (P_L \circ (K^T - K))) \\ &= \mathrm{tr}(Q [\bar{R} + \{P_L \circ (Q^T \bar{Q} - \bar{Q}^T Q + R \bar{R}^T - \bar{R} R^T)\} R^{+T}] \dot{A}^T) \\ &= \mathrm{tr}(\bar{A} \dot{A}^T) \; . \end{aligned}$$

In the above derivation the Lemmas B.0.19, B.0.18 and B.0.20 have been used.

□

The pullback can be computed in Taylor arithmetic. In the global derivative accumulation it is necessary to update the value of $[\bar{A}]_D$. This happens if $[A]_D$ is input of more than one function. The algorithm for the pullback takes this into consideration.

| | |
|---|---|
| **input** | : $[A]_D = [A_{[0]}, \ldots, A_{[D-1]}]$, where $A_{[d]} \in \mathbb{R}^{M \times N}$, $d = 0, \ldots, D-1$, $M \geq N$. |
| **input** | : $[Q]_D = [Q_{[0]}, \ldots, Q_{[D-1]}]$, where $Q_{[d]} \in \mathbb{R}^{M \times M}$ , $d = 0, \ldots, D-1$ |
| **input** | : $[R]_D = [R_{[0]}, \ldots, R_{[D-1]}]$, where $R_{[d]} \in \mathbb{R}^{M \times N}$ , $d = 0, \ldots, D-1$ |
| **input/output** | : $[\bar{A}]_D = [\bar{A}_{[0]}, \ldots, \bar{A}_{[D-1]}]$, where $\bar{A}_{[d]} \in \mathbb{R}^{M \times N}$, $d = 0, \ldots, D-1$, $M \geq N$. |
| **input** | : $[\bar{Q}]_D = [\bar{Q}_{[0]}, \ldots, \bar{Q}_{[D-1]}]$, where $\bar{Q}_{[d]} \in \mathbb{R}^{M \times M}$ , $d = 0, \ldots, D-1$ |
| **input** | : $[\bar{R}]_D = [\bar{R}_{[0]}, \ldots, \bar{R}_{[D-1]}]$, where $\bar{R}_{[d]} \in \mathbb{R}^{M \times N}$ , $d = 0, \ldots, D-1$ |

$$[\bar{A}]_D = [\bar{A}]_D + [Q]_D \left( [\bar{R}]_D + \left( P_L \circ \left( [R]_D [\bar{R}]_D^T - [\bar{R}]_D [R]_D^T + [Q]_D^T [\bar{Q}]_D - [\bar{Q}]_D^T [Q]_D \right) \right) [R]_D^{+T} \right)$$

**Algorithm 3.7.2:** Pullback of the full QR decomposition in Taylor arithmetic. The inputs $[A]_D, [Q]_D, [R]_D$ must satisfy the defining equations.

## 3.8. Thin QR Decomposition

Consider the linear least-squares problem $\min_x \|y - Ax\|^2$ with $A \in \mathbb{R}^{M\times N}$, $M \geq N$ and $\operatorname{rank}(A) = N$. In this case, the closed form solution $x = (A^TA)^{-1}Ay$ is known. If $A$ is ill-conditioned the multiplication $A^TA$ would square the condition number. It is possible to circumvent this multiplication using the QR decomposition: First, compute $A = QR$ and then solve $R^TRx = Ay$ by a forward resp. back substitution. That means by using the QR decomposition the matrix-matrix product can be avoided. The thin QR decomposition is also useful in a variety of other situations.

**Definition 3.8.1** (thin QR decomposition)**.** Let $A \in \mathbb{R}^{M\times N}$. Then the factorization $A = QR$ is called the *thin* QR decomposition of $A$ if

$$\begin{aligned} 0 =& QR - A \\ 0 =& Q^TQ - \mathbf{1}_K \\ 0 =& P_L \circ R \end{aligned}$$

is satisfied, where $Q \in \mathbb{R}^{M\times K}$, $R \in \mathbb{R}^{K\times N}$ and $K := \min(M, N)$. The functional dependence is denoted

$$Q, R = \operatorname{qr}(A)\ . \tag{3.33}$$

**Proposition 3.8.1** (uniqueness of the thin QR decomposition)**.** *Let $A \in \mathbb{R}^{M\times N}$ s.t. $\operatorname{rank}(A) = N$ , $R \in \mathbb{R}^{N\times N}$ and $Q \in \mathbb{R}^{M\times N}$ be given. Then the thin QR decomposition is unique if the diagonal elements of $R$ are chosen positive, i.e., $R_{nn} > 0$ for all $n = 1, \ldots, N$.*

*Proof.* Assume there are two different thin QR decompositions $A = Q_1R_1 = Q_2R_2$. One concludes from the uniqueness of the Cholesky decomposition that $A^TA = R_1^TR_1 = R_2^TR_2$ implies $R_1 = R_2$. The uniqueness of $Q$ follows from $Q_1 = AR_1^{-1} = AR_2^{-1} = Q_2$. Stewart [1998] □

### 3.8.1. Pushforward

Again, the approach is to apply Newton-Hensel lifting to the defining equations.

**Proposition 3.8.2** (Newton-Hensel lifting of the thin QR decomposition)**.** *Let $[A]_{D+E} \in \mathbb{R}[T]/(T^D)^{M\times N}$ with $M \geq N$ and $1 \leq E \leq D$, $[R]_D \in \mathbb{R}[T]/(T^D)^{N\times N}$ upper triangular with nonsingular $R_{[0]}$ and $[Q]_D \in \mathbb{R}[T]/(T^D)^{M\times N}$ be given and satisfy the defining equations of order $D$. Then $\Delta R \equiv [R]_{D:D+E}$ and $\Delta Q \equiv [Q]_{D:D+E}$ are given by*

$$\begin{aligned} [\Delta F]_E T^{D\overset{D+E}{=}} & - [Q]_D[R]_D + [A]_{D+E} \\ [\Delta G]_E T^{D\overset{D+E}{=}} & - [Q]_D^T[Q]_D + \mathbf{1} \\ [S]_E &\overset{E}{=} \frac{1}{2}[\Delta G]_E \\ P_L \circ ([X]_E) &\overset{E}{=} P_L \circ \left([Q]_E^T[\Delta F]_E[R]_E^{-1}\right) - P_L \circ [S]_E \\ [\Delta R]_E &\overset{E}{=} [Q]_E^T[\Delta F]_E - ([S]_E + [X]_E)[R]_E \\ [\Delta Q]_E &\overset{E}{=} \left([\Delta F]_E - [Q]_E[\Delta R]_E\right)[R]_E^{-1}\ . \end{aligned}$$

*Proof.* Look at the first defining equation

$$\begin{aligned}
0 &\overset{D+E}{=} [Q]_{D+E}[R]_{D+E} - [A]_{D+E} \\
&\overset{D+E}{=} ([Q]_D + [\Delta Q]_E T^D)([R]_D + [\Delta R]_E T^D) - [A]_{D+E} \\
&\overset{D+E}{=} [Q]_D[R]_D - [A]_{D+E} + ([\Delta Q]_E[R]_D + [Q]_D[\Delta R]_E)T^D \\
&\overset{D+E}{=} -[\Delta F]_E T^D + ([\Delta Q]_E[R]_D + [Q]_D[\Delta R]_E)T^D \\
&\overset{E}{=} -[\Delta F]_E + [\Delta Q]_E[R]_E + [Q]_E[\Delta R]_E \,. \qquad (3.34)
\end{aligned}$$

where the known and the unknown parts get separated. Similarly for the second defining equation

$$\begin{aligned}
0 &\overset{D+E}{=} [Q]^T_{D+E}[Q]_{D+E} - \mathbf{1} \\
0 &\overset{D+E}{=} [Q]^T_D[Q]_D - \mathbf{1} + ([Q]^T_D[\Delta Q]_E + [\Delta Q]^T_E[Q]_D)T^D \\
0 &\overset{E}{=} -[\Delta G]_E + [Q]^T_E[\Delta Q]_E + [\Delta Q]^T_E[Q]_E \\
0 &= -[\Delta G]_E + [S]_E + [X] + [S]_E - [X] \\
\therefore \quad S &= \frac{1}{2}[\Delta G]_E \,,
\end{aligned}$$

where it has been used that every matrix can be written as the sum of a symmetric and an antisymmetric matrix (Lemma B.0.12). Now multiply (3.34) by $[Q]^T_E$ from the left to obtain

$$0 \overset{E}{=} -[Q]^T_E[\Delta F]_E + [Q]^T_E[\Delta Q]_E[R]_E + [\Delta R]_E \qquad (3.35)$$

$$0 \overset{E}{=} -[Q]^T_E[\Delta F]_E + ([S]_E + [X]_E)[R]_E + [\Delta R]_E \qquad (3.36)$$

$$\therefore \quad P_L \circ [X]_E \overset{E}{=} P_L \circ \left([Q]^T_E[\Delta F]_E[R]^{-1}_E - [S]_E\right) \,.$$

Since $[X]_E$ is antisymmetric it is fully defined by $P_L \circ [X]_E$. From (3.35) it follows

$$[\Delta R]_E \overset{E}{=} [Q]^T_E[\Delta F]_E - ([S]_E + [X]_E)[R]_E$$

and from (3.34) multiplied from the right by $[R]^{-1}_E$ one obtains

$$[\Delta Q]_E \overset{E}{=} ([\Delta F]_E - [Q]_E[\Delta R]_E)\,[R]^{-1}_E \,.$$

□

**input** : $[A]_D = [A_{[0]}, \ldots, A_{[D-1]}]$, where $A_{[d]} \in \mathbb{R}^{M \times N}$, $d = 0, \ldots, D-1$, $M \geq N$, $A_{[0]}$ has full column-rank
**output**: $[Q]_D = [Q_{[0]}, \ldots, Q_{[D-1]}]$ matrix with orthonormal column vectors, where $Q_{[d]} \in \mathbb{R}^{M \times N}$ , $d = 0, \ldots, D-1$
**output**: $[R]_D = [R_{[0]}, \ldots, R_{[D-1]}]$ upper triangular, where $R_{[d]} \in \mathbb{R}^{N \times N}$ , $d = 0, \ldots, D-1$

$Q_{[0]}, R_{[0]} = \mathrm{qr}(A_{[0]})$

**for** $d = 1$ **to** $D-1$ **do**

$\Delta F = A_{[d]} - \sum_{k=1}^{d-1} Q_{[d-k]} R_{[k]}$
$S = -\frac{1}{2} \sum_{k=1}^{d-1} Q_{[d-k]}^T Q_{[k]}$
$P_L \circ X = P_L \circ (Q_{[0]}^T \Delta F R_{[0]}^{-1} - S)$
$X = P_L \circ X - (P_L \circ X)^T$
$R_{[d]} = Q_{[0]}^T \Delta F - (S + X) R_{[0]}$
$Q_{[d]} = (\Delta F - Q_{[0]} R_{[d]}) R_{[0]}^{-1}$

**end**

**Algorithm 3.8.1:** Sequential Hensel lifting of the thin QR decomposition for $A \in \mathbb{R}[T]/(T^D)^{M \times N}$ assuming $M \geq N$. This algorithm shows the special case $E = 1$ of the Proposition 3.8.2.

### 3.8.2. Pullback

**Proposition 3.8.3** (pullback of the thin QR decomposition)**.** *Let $A \in \mathbb{R}^{M \times N}$ with $M \geq N$ and full column rank be given. Then the pullback can be written in a single equation*

$$\bar{A} = Q \left( \bar{R} + P_L \circ \left( R \bar{R}^T - \bar{R} R^T + Q^T \bar{Q} - \bar{Q}^T Q \right) R^{-T} \right) + (\bar{Q} - Q Q^T \bar{Q}) R^{-T} \,. \tag{3.37}$$

*The last term drops out for square $A \in \mathbb{R}^{N \times N}$.*

*Proof.* A perturbation of $A$ by $\dot{A}$ in the implicit system

$$\begin{aligned} 0 &= A - QR \\ 0 &= Q^T Q - \mathbf{1} \\ 0 &= P_L \circ R \end{aligned}$$

results, to first order, in

$$\begin{aligned} 0 &= \dot{A} - \dot{Q} R - Q \dot{R} && (*) \\ 0 &= \dot{Q}^T Q + Q^T \dot{Q} \quad . && (**) \end{aligned}$$

Define the antisymmetric matrix $X := Q^T \dot{Q}$. Transforming (*) by left multiplication with $Q^T$ and right multiplication with $R^{-1}$ yields

$$\begin{aligned} 0 &= Q^T \dot{A} R^{-1} - Q^T Q \dot{R} R^{-1} - Q^T \dot{Q} \\ \text{therefore} \quad P_L \circ X &= P_L \circ (Q^T \dot{A} R^{-1}) \\ \text{and thus} \quad X &= P_L \circ X - (P_L \circ X)^T \,. \end{aligned}$$

Multiplying $Q^T$ from the left to (*) yields $\dot{R} = Q^T \dot{A} - XR$ and multiplying $R^{-1}$ from the right

to (*) leads to $\dot{Q} = (\dot{A} - Q\dot{R})R^{-1}$. Thus, one is now in place to calculate

$$\begin{aligned}
&\operatorname{tr}(\bar{Q}^T\dot{Q}) + \operatorname{tr}(\bar{R}^T\dot{R}) = \operatorname{tr}(\bar{Q}^T(\dot{A} - Q\dot{R})R^{-1}) + \operatorname{tr}(\bar{R}^T\dot{R}) \\
=&\operatorname{tr}(R^{-1}\bar{Q}^T\dot{A}) + \operatorname{tr}(\underbrace{(\bar{R}^T - R^{-1}\bar{Q}^TQ)}_{=:F}\dot{R}) \\
=&\operatorname{tr}(R^{-1}\bar{Q}^T\dot{A}) + \operatorname{tr}(F(Q^T\dot{A} - XR)) \\
=&\operatorname{tr}((R^{-1}\bar{Q}^T + FQ^T)\dot{A}) + \operatorname{tr}(-RFX) \\
=&\operatorname{tr}((R^{-1}\bar{Q}^T + FQ^T)\dot{A}) + \operatorname{tr}(-RF(P_L \circ X - (P_L \circ X)^T)) \\
=&\operatorname{tr}((R^{-1}\bar{Q}^T + FQ^T)\dot{A}) + \operatorname{tr}(-(RF)(P_L \circ X)) + \operatorname{tr}((RF)^T(P_L \circ X)) \\
=&\operatorname{tr}((R^{-1}\bar{Q}^T + FQ^T)\dot{A}) + \operatorname{tr}(-X^T(P_L \circ (RF)^T)) + \operatorname{tr}(X^T(P_L \circ (RF))) \\
=&\operatorname{tr}((\bar{Q}R^{-T} + QF^T)\dot{A}^T) + \operatorname{tr}(R^{-T}\dot{A}^TQ(P_L \circ (RF - F^TR^T)) \\
=&\operatorname{tr}((QF^T + \bar{Q}R^{-T} + QP_L \circ (RF - F^TR^T)R^{-T})\dot{A}^T)
\end{aligned}$$

In the above derivation the Lemmas B.0.19, B.0.18 and B.0.20 have been used. From the last equation follows that

$$\bar{A} = Q\left(\bar{R} + \left(P_L \circ (Q^T\bar{Q} - \bar{Q}^TQ + R\bar{R}^T - \bar{R}R^T)\right)R^{-T}\right) + (\bar{Q} - QQ^T\bar{Q})R^{-T} .$$

□

### 3.8.3. Wide QR Decomposition

Above, only algorithms for tall matrices $A \in \mathbb{R}^{M\times N}$, $M \geq N$ haven derived. The case $M < N$ can be treated as follows: the matrix $A$ is split into a square matrix $A_1 \in \mathbb{R}^{M\times M}$ and a matrix $A_2 \in \mathbb{R}^{M\times N-M}$ s.t. $A = (A_1, A_2)$. Then the QR decomposition of $A_1$ is performed, i.e., $QR_1 = A_1$. Then $R_2$ in $R = (R_1, R_2)$ is computed as $R_2 = Q^TA_2$. Hence, $Q$ and $R$ satisfy $QR = A$. The computational graph of these operations is depicted in Figure 3.3.

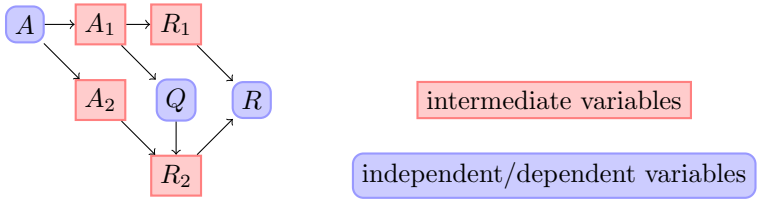



Figure 3.3.: This graph explains how the QR decomposition of a $A \in \mathbb{R}^{M\times N}$ matrix with $M < N$ can be computed.

**Proposition 3.8.4** (Pullback of the QR decomposition for $M < N$)**.** *It is necessary to perform the pullback of the computational graph as depicted in Figure 3.3. Given $\bar{R} \in \mathbb{R}^{M\times N}$ and $\bar{Q} \in \mathbb{R}^{M\times M}$, then $\bar{A}_2 \in \mathbb{R}^{N-M\times M}$ and $\bar{Q} \in \mathbb{R}^{M\times M}$ are incremented as*

$$\bar{A}_2 = \bar{A}_2 + Q\bar{R}_2 \tag{3.38}$$
$$\bar{Q} = \bar{Q} + A_2\bar{R}_2^T . \tag{3.39}$$

*Proof.*

$$\begin{aligned} \dot{R}_2 =& Q^T \dot{A}_2 + \dot{Q}^T A_2 \\ \therefore \quad \mathrm{tr}(\bar{R}_2^T \dot{R}_2) =& \mathrm{tr}(\bar{R}_2^T (Q^T \dot{A}_2 + \dot{Q}^T A_2)) \\ =& \mathrm{tr}(\bar{R}_2^T Q^T \dot{A}_2) + \mathrm{tr}(A_2 \bar{R}_2^T \dot{Q}^T) \ . \end{aligned}$$

□

**input** : $[A]_D = [A_{[0]}, \ldots, A_{[D-1]}]$, where $A_{[d]} \in \mathbb{R}^{M \times N}$, $d = 0, \ldots, D-1$, $M \geq N$.
**input** : $[Q]_D = [Q_{[0]}, \ldots, Q_{[D-1]}]$ matrix with orthonormal column vectors, where $Q_{[d]} \in \mathbb{R}^{M \times N}$ , $d = 0, \ldots, D-1$
**input** : $[R]_D = [R_{[0]}, \ldots, R_{[D-1]}]$ upper triangular, where $R_{[d]} \in \mathbb{R}^{N \times N}$ , $d = 0, \ldots, D-1$
**input/output**: $[\bar{A}]_D = [\bar{A}_{[0]}, \ldots, \bar{A}_{[D-1]}]$, where $\bar{A}_{[d]} \in \mathbb{R}^{M \times N}$, $d = 0, \ldots, D-1$, $M \geq N$.
**input** : $[\bar{Q}]_D = [\bar{Q}_{[0]}, \ldots, \bar{Q}_{[D-1]}]$, where $\bar{Q}_{[d]} \in \mathbb{R}^{M \times N}$ , $d = 0, \ldots, D-1$
**input** : $[\bar{R}]_D = [\bar{R}_{[0]}, \ldots, \bar{R}_{[D-1]}]$, where $\bar{R}_{[d]} \in \mathbb{R}^{N \times N}$ , $d = 0, \ldots, D-1$

$$\begin{aligned} [\bar{A}]_D =& [\bar{A}]_D + ([\bar{Q}]_D - [Q]_D [Q]_D^T [\bar{Q}]_D)[R]_D^{-T}) \\ &+ [Q]_D \left( [\bar{R}]_D + P_L \circ \left( [R]_D [\bar{R}]_D^T - [\bar{R}]_D [R]_D^T + [Q]_D^T [\bar{Q}]_D - [\bar{Q}]_D^T [Q]_D \right) [R]_D^{-T} \right) \end{aligned}$$

**Algorithm 3.8.2:** Pullback of the thin QR decomposition with $A$ full column rank.

## 3.9. Real Symmetric Eigenvalue Decomposition

The problem of finding eigenvalues and eigenvectors arises in a wide variety of practical applications. It is desired to have algorithms that compute the real symmetric eigenvalue decomposition in univariate Taylor polynomial arithmetic as well as pullback algorithms. For instance, to minimize the largest eigenvalue based on a smooth relaxation of the problem such derivatives can be useful, see for instance Xin et al. [2004]. The symmetric eigenvalue decomposition is also important since the singular value decomposition (SVD) of real matrices is closely related to it. More explicitly, let $A \in \mathbb{R}^{N\times N}$ have rank $r$. One can compute the SVD $A = U\Sigma V^T$ and obtains the factors $\Sigma = \mathrm{diag}(\Sigma_1, 0)$, $U = (U_1, U_2)$, $U_1 \in \mathbb{R}^{M\times r}$, $V = (V_1, V_2)$, $V_1 \in \mathbb{R}^{N\times r}$. Alternatively, if one populates a matrix $C$ and then performs a real symmetric eigenvalue decomposition, i.e.,

$$C = \begin{pmatrix} 0 & A \\ A^T & 0 \end{pmatrix} = P^T \begin{pmatrix} \Sigma_1 & 0 & 0 \\ 0 & -\Sigma_1 & 0 \\ 0 & 0 & 0 \end{pmatrix} P \,,$$

then one can find, according to Björck [1996], that the orthogonal matrix $P$ has the structure

$$P = \frac{1}{\sqrt{2}} \begin{pmatrix} U_1 & U_1 & \sqrt{2}U_2 & 0 \\ V_1 & -V_1 & 0 & \sqrt{2}V_2 \end{pmatrix}^T \,.$$

**Definition 3.9.1** (real symmetric eigenvalue decomposition)**.** Let $A \in \mathbb{R}^{N\times N}$ be symmetric. Then the factorization $A = Q\Lambda Q^T$ is called the (real) symmetric eigenvalue decomposition if the matrices $A, Q, \Lambda \in \mathbb{R}^{N\times N}$ satisfy

$$\begin{aligned} 0 =& Q^T A Q - \Lambda \\ 0 =& Q^T Q - \mathbf{1}_M \\ 0 =& (P_L + P_R) \circ \Lambda \,, \end{aligned}$$

where $P_L$ and $P_R$ are skeletal projectors (Definition 3.3.3).

### 3.9.1. Pushforward

Consider the univariate Taylor polynomial with symmetric matrices as elements $[A]_D \in (\mathbb{R}[T]/(T^D))^{N\times N}$. In [Kato, 1966, Chapter II, Theorem 6.1] it is shown that when a holomorphic family $A(t)$ is symmetric for all $t \in (-\epsilon, \epsilon)$, then the eigenvalues $\lambda_m(t)$ and eigenprojections $P_m(t)$ are holomorphic on the real axis, where $m = 1, \ldots, M$ and $M \leq N$ denotes the number of distinct eigenvalues. A discussion on the existence of analytical paths can also be found in [Andrew and Tan, 1998] where an algorithm for the generalized eigenvalue problem $A(t)X(t) = \Lambda(t)B(t)X(t)$ is presented for $A(t), B(t)$ hermitian and analytic on an open neighborhood of $t \in (-\epsilon, \epsilon)$. Also noteworthy is the paper by van der Aa et al. [2007] where the generalized eigenvalue problem is treated when $A(t)$ is a general non-defective complex matrix and $B(t)$ the identity matrix.

The eigenvalue decomposition in univariate Taylor arithmetic is the solution $[\Lambda]_D, [Q]_D \in \mathbb{R}[T]/(T^D)^{N\times N}$ of the implicit system

$$0 \overset{D}{=} [Q]_D^T [A]_D [Q]_D - [\Lambda]_D \tag{3.40}$$

$$0 \overset{D}{=} [Q]_D^T [Q]_D - \mathbf{1} \tag{3.41}$$

$$0 \overset{D}{=} (P_L + P_R) \circ [\Lambda]_D \,. \tag{3.42}$$

These equations are called the *defining equations of order D*. The functional dependence is

denoted

$$[\Lambda]_D, [Q]_D = \text{eigh}([A]_D) \ . \tag{3.43}$$

Let $\Lambda, Q = \text{eigh}(A)$ be the usual symmetric eigenvalue decomposition. The diagonal of $[\Lambda]_D$ is denoted by the one-dimensional vector polynomial

$$[\lambda]_D = \text{diag}([\Lambda]_D) \in \mathbb{R}[T](T^D)^N \ .$$

If eigenvalues are repeated, i.e., multiple, the eigenvectors generalize to eigenspaces and the columns of $Q$, that are associated to such a multiple eigenvalue, are not unique. Rather, any orthonormal basis could be the result. This has consequences for the Hensel-Newton lifting approach. I.e., when $[Q]_D$ and $[R]_D$ satisfy the defining equations of order $D$ it may be impossible to find a $[\Delta Q]_E$ and $[\Delta R]_E$ such that $[Q]_{D+E} = [Q]_D + [\Delta Q]_E T^D$ and $[R]_{D+E} = [R]_D + [\Delta R]_E T^D$ satisfy the defining equations of order $D+E$. The higher-order coefficients $[\Delta A]_E$ enforce additional conditions on the chosen basis of the eigenspaces. A wrong choice of $[Q]_D$ means that $0 \overset{D+E}{=} (P_L + P_R) \circ [\Lambda]_{D+E}$ cannot be satisfied. On the other hand, the relaxed condition $0 \overset{D}{=} P_b^D \circ [\Lambda]_{D+E}$ can be still satisfied. The matrix $P_b^D$ is a skeletal projector with zero blocks on the main diagonal whose size corresponds to the multiplicity of an eigenvalue $[\lambda]_D$ and all other entries are ones. In general, the *multiplicity*

$$m^d([\lambda_j]_D)$$

of an eigenvalue $[\lambda_j]_D$ of *level* $d$ is defined to be the number of $i \in \mathbb{N}$ s.t. $[\lambda_j]_D \overset{d}{=} [\lambda_i]_D$. I.e.,

$$\text{diag}([\Lambda]_d) = (\underbrace{[\lambda_1]_d, \ldots, [\lambda_1]_d}_{m^d([\lambda_1]_D) \text{ times}}, \ldots, \underbrace{[\lambda_{{N_\text{b}}^d}]_d, \ldots, [\lambda_{{N_\text{b}}^d}]_d}_{m^d([\lambda_{{N_\text{b}}^d}]_D) \text{ times}}),$$

where ${N_\text{b}}^d$ is the number of different eigenvalues at level $d$. Let $b^d \in \mathbb{N}^{{N_\text{b}}^d+1}$ be a vector satisfying

$$m^d([\lambda_{n_\text{b}}]_D) = b^d_{n_\text{b}+1} - b^d_{n_\text{b}} \ .$$

The symbol $b$ is used because it relates to blocks in the matrix. The elements of $P_b^d$ satisfy $(P_b^d)_{ij} = 1 - \sum_{n_\text{b}=1}^{{N_\text{b}}^d+1} \delta_{b^d_{n_\text{b}} \leq i < b^d_{n_\text{b}+1}} \delta_{b^d_{n_\text{b}} \leq j < b^d_{n_\text{b}+1}}$. This notation is a little cumbersome but turns out to be helpful. One defines $b^0 := [1, N+1]$. The vector $b^1$ represents the multiplicities of the usual symmetric eigenvalue decomposition and for $N = 3$ and $b^d = [1, 3, 4]$ one would have

$$P_b^d = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} \ .$$

The overall problem is reformulated a sequence of subproblems which are easier to solve. As already stated above, the idea is to relax the problem and then define an iteration which converges to the desired result:

1. Solve relaxed problem of level 1 and order $D$
2. Solve relaxed problem of level 2 and order $D$
3. etc.

**Definition 3.9.2** (relaxed problem of level $d$ and order $D$)**.** The implicit system

$$\begin{aligned}
0 &\overset{D}{=} [Q^d]_D^T [A]_D [Q^d]_D - [\Lambda^d]_D \\
0 &\overset{D}{=} [Q^d]_D^T [Q^d]_D - \mathbf{I} \\
0 &\overset{d}{=} (P_L + P_R) \circ [\Lambda]_d \\
0 &\overset{D}{=} P_b^d \circ [\Lambda^d]_D \ ,
\end{aligned}$$

is called the *relaxed problem of level $d$ and order $D$*. I.e., it is assumed that up to order $d$ the original problem is solved but only block diagonalized for the higher order coefficients.

To give an illustrative example consider this relaxed problem of order 3 and level 2. At this point of the algorithm, one has potentially obtained a matrix polynomial $[\Lambda]_3 = \sum_{d=0}^{2} \Lambda_{[d]} T^d$ with coefficients of the form

$$\Lambda_{[0]} = \begin{pmatrix} 3 & & & & & & \\ & 3 & & & & & \\ & & 3 & & & & \\ & & & 2 & & & \\ & & & & 2 & & \\ & & & & & 1 & \end{pmatrix}, \quad \Lambda_{[1]} = \begin{pmatrix} 2 & & & & & & \\ & 2 & & & & & \\ & & 3 & & & & \\ & & & 2 & & & \\ & & & & 2 & & \\ & & & & & 2 & \end{pmatrix}, \quad \Lambda_{[2]} = \begin{pmatrix} 1 & 3 & & & & & \\ 3 & 5 & & & & & \\ & & 7 & & & & \\ & & & 1 & 2 & & \\ & & & 2 & 3 & & \\ & & & & & 7 & \end{pmatrix}.$$

I.e., $\Lambda_{[0]}$ and $\Lambda_{[1]}$ are already diagonal. Since there are two eigenvalues with multiplicity $m^2([\lambda]_3) = 2$ it follows that $\Lambda_{[2]}$ is only block diagonal. Note that the eigenvalues are not globally sorted by value in the higher coefficients but only in the subblocks defined by the lower order coefficients. In this example, the repeated eigenvalues in the first block split at the lift from $d = 0$ to $d = 1$. The blocks are defined by $b^1 = [1, 4, 6, 7]$ and $b^2 = [1, 3, 4, 6, 7]$. The blocks in $\Lambda_2$ are defined by $b^2$.

The function that solves the relaxed problem of order $D$ and level $d$ is denoted

$$[\Lambda^d]_D, [Q^d]_D = \underset{d}{\text{eigh}}([A]_D) \ . \tag{3.44}$$

The idea is to implement an algorithm that successively increases $d$ by one. For consistency one defines $[Q^0]_D := \mathbf{I}$ and $[\Lambda^0]_D := [A]_D$.

**Theorem 3.9.1.** *Let $[A]_D$ be given, then the solution of*

$$[Q^{d+1}]_D, [\Lambda^{d+1}]_D \overset{D}{=} \underset{d+1}{\text{eigh}}([A]_D)$$

*can be computed from the solution $[Q^d]_D, [\Lambda^d]_D \overset{D}{=} \text{eigh}_d([A]_D)$ by computing*

$$[\hat{\Lambda}_{s,s}]_{D-d}, [\hat{Q}_{s,s}]_{D-d} \overset{D-d}{=} \underset{1}{\text{eigh}}([\Lambda^d_{s,s}]_{d:}) \ , \tag{3.45}$$

*where $s = b^d_{n_\text{b}} : b^d_{n_\text{b}+1} - 1$ are slice indices and $n_\text{b} = 1, \ldots, {N_\text{b}}^d$. All other elements of $[\hat{Q}]_{D-d}$ and $[\hat{\Lambda}]_{D-d}$ are zero. I.e., $[\hat{Q}]_{D-d}$ and $[\hat{\Lambda}]_{D-d}$ are block diagonal. It holds that*

$$\begin{aligned}
[\Lambda^{d+1}]_D &\overset{D}{=} [\Lambda^d]_d + [\hat{\Lambda}]_{D-d} T^d \\
[Q^{d+1}]_D &\overset{D}{=} [Q^d]_D [Q]_D \ ,
\end{aligned}$$

*where* $[Q]_D = [\hat{Q}]_{D-d} + [\Delta Q]_d T^{D-d}$ *for some* $[\Delta Q]_{D-d}$ *that satisfies*

$$0 \overset{D}{=} [Q]_D^T [Q]_D - \mathbf{1} \,. \tag{3.46}$$

*Proof.* It is necessary to show that $[\Lambda^{d+1}]_D$, $[Q^{d+1}]_D$ is a solution to the relaxed equations of level $d+1$ and order $D$. From the definition of $\text{eigh}_1$ it follows that $0 = (P_L + P_R) \circ [\Lambda^{d+1}]_{d+1}$ and $0 = P_b^{d+1} \circ [\Lambda^{d+1}]_D$ is satisfied. It is also known that $0 \overset{D}{=} [Q^{d+1}]_D^T [Q^{d+1}]_D - \mathbf{1} \overset{D}{=} [Q]_D^T [Q^d]_D^T [Q^d]_D [Q]_D - \mathbf{1}$ is satisfied because $0 \overset{D}{=} [Q^d]_D^T [Q^d]_D - \mathbf{1}$ and $0 \overset{D}{=} [Q]_D^T [Q]_D^T - \mathbf{1}$. Hence, it only remains to show that the third defining equation is satisfied which is shown by the following straight-forward calculation:

$$\begin{aligned}
0 &\overset{D}{=} [Q]_D^T [Q^d]_D^T [A]_D [Q^d]_D [Q]_D - [\Lambda^{d+1}]_D \\
&\overset{D}{=} [Q]_D^T [\Lambda^d]_D [Q]_D - [\Lambda^{d+1}]_D \\
&\overset{D}{=} [Q]_D^T ([\Lambda^d]_d + [\Lambda^d]_{d:} T^d) [Q]_D - [\Lambda^d]_d - [\hat{\Lambda}]_{D-d} T^d \\
&\overset{D}{=} [Q]_D^T [\Lambda^d]_d [Q]_D + [Q]_D^T [\Lambda^d]_{d:} [Q]_D T^d - [\Lambda^d]_d - [\hat{\Lambda}]_{D-d} T^d \\
&\overset{D}{=} [\Lambda^d]_d [Q]_D^T [Q]_D + [\hat{Q}]_{D-d}^T [\Lambda^d]_{d:} [\hat{Q}]_{D-d} T^d - [\Lambda^d]_d - [\hat{\Lambda}]_{D-d} T^d \\
&\overset{D}{=} [\hat{Q}]_{D-d}^T [\Lambda^d]_{d:} [\hat{Q}]_{D-d} T^d - [\hat{\Lambda}]_{D-d} T^d \\
&\overset{D-d}{=} [\hat{Q}]_{D-d}^T [\Lambda^d]_{d:} [\hat{Q}]_{D-d} - [\hat{\Lambda}]_{D-d} \,.
\end{aligned}$$

In the fifth line it has been used that the diagonalization has only to be performed for block diagonal matrices. If the eigenvalues are already distinct there is nothing to diagonalize and the step can be skipped. It also means that one may interchange $[\Lambda^d]_d$ with $[Q]_D$. □

The following proposition provides the means to diagonalize a matrix in the zeroth degree and block diagonalize w.r.t. the blocks defined by the repeated eigenvalues. I.e., it gives the justification that the solution of (3.45) can be found. In the case of distinct eigenvalues the application of this algorithm already solves the original problem.

**Proposition 3.9.2.** *Let* $[A]_{D+E} = [A]_D + [\Delta A]_E T^D \in \mathbb{R}[T]/(T^D)^{N \times N}$ *be given and* $[\Lambda^d]_D$, $[Q^d]_D$ *be a solution of the relaxed problem of level* $d = 1$ *and order* $D$*. Then there exist* $[\Delta\Lambda^d]_E$ *and* $[\Delta Q^d]_E$ *such that* $[\Lambda^d]_{D+E} = [\Lambda^d]_D + [\Delta\Lambda^d]_E T^D$ *and* $[\Delta Q^d]_{D+E} = [\Delta Q^d]_D + [\Delta Q^d]_E T^D$ *are a solution of the relaxed problem of level* $d = 1$ *and order* $D + E$*. A closed form solution is*

$$[\Delta\Lambda^d]_E \overset{E}{=} \bar{P}_b^d \circ [K]_E \tag{3.47}$$

$$[\Delta Q^d]_E \overset{E}{=} [Q^d]_E \left( [\Delta G]_E + P_b^d \circ ([K]_E / [E]_E) \right) \tag{3.48}$$

*where*

$$\begin{aligned}
[\Delta F]_E T^D &\overset{D+E}{=} [Q^d]_D^T [A]_D [Q^d]_D - [\Lambda^d]_D \\
[\Delta G]_E T^D &\overset{D+E}{=} -\frac{1}{2} \left( [Q^d]_D^T [Q^d]_D - \mathbf{1} \right)
\end{aligned}$$

*and*

$$\begin{aligned}
[K]_E &\overset{E}{=} [\Delta F]_E + ([\Lambda]_E [\Delta G]_E + [\Delta G]_E [\Lambda]_E) + [Q^d]_E^T [\Delta A]_E [Q^d]_E \\
[E_{ij}]_E &\overset{E}{=} [\Lambda_{jj}^d]_E - [\Lambda_{ii}^d]_E \,.
\end{aligned}$$

*The expression* $[K]_E / [E]_E$ *denotes an element-wise division.* $P_b^d$ *is a matrix with only ones on the diagonal blocks defined by the multiplicity of eigenvalues in* $\Lambda_{[0]}$*.* $\bar{P}_b^d$ *is defined s.t.* $\bar{P}_b^d + P_b^d$

*is a matrix full of ones. One can see here that if the eigenvalues are distinct, then* $\bar{P}_b^d$ *is the identity matrix* $\mathbf{1}$*.*

*Proof.* Set $Q^d \equiv Q$ etc. for notational simplicity. Applying Newton-Hensel lifting to the defining equations yields

$$\begin{aligned}
0 &\overset{D+E}{=} ([Q]_D + [\Delta Q]_E T^D)^T ([Q]_D + [\Delta Q]_E T^D) - \mathbf{1} \\
&\overset{E}{=} -2[\Delta G]_E + [\Delta Q]_E^T [Q]_E + [Q]_E^T [\Delta Q]_E \\
&\overset{E}{=} -2[\Delta G]_E + 2[S]_E \,, \\
0 &\overset{D+E}{=} ([Q]_D + [\Delta Q]_E T^D)^T ([A]_D + [\Delta A]_E T^D)([Q]_D + [\Delta Q]_E T^D) - ([\Lambda]_D + [\Delta\Lambda]_E T^D) \\
&\overset{E}{=} [\Delta F]_E + [Q]_E^T [\Delta A]_E [Q]_E + [\Delta Q]_E^T [Q]_E [\Lambda]_E + [\Lambda]_E [Q]_E^T [\Delta Q]_E - [\Delta\Lambda]_E \\
&\overset{E}{=} [K]_E + [X]_E [\Lambda]_E - [\Lambda]_E [X]_E - [\Delta\Lambda]_E \\
&\overset{E}{=} [K]_E + [E]_E \circ [X]_E - [\Delta\Lambda]_E \,. \qquad (3.49)
\end{aligned}$$

Thus $[\Delta\Lambda]_E \overset{E}{=} \bar{P}_b^d \circ [K]_E$ and $[X]_E^T \overset{E}{=} P_b^d \circ ([K]_E / [E]_E)$. Above, $[\Delta Q]_E^T [Q]_E \overset{E}{=} [S]_E + [X]_E$, $[S]_E$ symmetric and $[X]_E$ antisymmetric (Lemma B.0.12) has been used. □

It remains to show that (3.46) can be satisfied.

**Lemma 3.9.3.** *Let* $[Q]_D$ *be given and it satisfies the defining equation* $0 \overset{D}{=} [Q]_D^T [Q]_D - \mathbf{1}$*. Then the solution can be lifted to* $D+E$ *with* $E \leq D$*. I.e., it is possible to find* $[Q]_{D+E} := [Q]_D + [\Delta Q]_E T^D$ *s.t.* $0 \overset{D+E}{=} [Q]_{D+E}^T [Q]_{D+E} - \mathbf{1}$*. A closed form solution for* $[\Delta Q]_E$ *is given by*

$$[\Delta Q]_E \overset{E}{=} [Q]_E [S]_E \,, \qquad (3.50)$$

*where* $[S]_E T^D \overset{D+E}{=} -\frac{1}{2}\left([Q]_D^T [Q]_D - \mathbf{1}\right)$*.*

*Proof.*

$$\begin{aligned}
0 &\overset{D+E}{=} ([Q]_D + [\Delta Q]_E T^D)^T ([Q]_D + [\Delta Q]_E T^D) - \mathbf{1} \\
&\overset{D+E}{=} ([Q]_D^T [Q]_D - \mathbf{1}) + ([Q]_D^T [\Delta Q]_E + [\Delta Q]_E^T [Q]_D) T^D \\
&\overset{E}{=} [\Delta G]_E + [Q]_E^T [\Delta Q]_E + [\Delta Q]_E^T [Q]_E \\
&\overset{E}{=} [\Delta G]_E + 2[S]_E \\
[\Delta Q]_E &\overset{E}{=} -\frac{1}{2}[Q]_E [\Delta G]_E \,,
\end{aligned}$$

where $[\Delta Q]_E^T [Q]_E = [S]_E + [X]_E$, $[S]_E$ symmetric and $[X]_E$ antisymmetric and $[\Delta G_E]_E T^D \overset{D+E}{=} \left(Q^T Q - \mathbf{1}\right)$. Since no condition defines constraints on $[X]_E$ it has been set to zero. □

**Example 3.9.1.** Let $f : \mathbb{R}^2 \to \mathbb{R}$ be defined by

$$\begin{aligned}
f(x) &= \lambda_1(A(x)) \\
A(x) &= \begin{pmatrix} x_1 & x_2 \\ x_2 & -x_1 \end{pmatrix} \,.
\end{aligned}$$

**input** : $[Q]_d = [Q_{[0]}, \ldots, Q_{[d-1]}]$ with $0 \overset{d}{=} [Q]_d^T [Q]_d - \mathbf{1}$
**input** : $D \in \mathbb{N}$
**output**: $[Q]_D = [Q_{[0]}, \ldots, Q_{[D-1]}]$, where $0 \overset{D}{=} [Q]_D^T [Q]_D - \mathbf{1}$

**for** $k = d$ **to** $D - 1$ **do**
  $Q_{[k]} = -\frac{1}{2} Q_{[0]} \sum_{i=1}^{k-1} Q_{[i]}^T Q_{[k-i]}$
**end**

**Algorithm 3.9.1:** This algorithm computes $[Q]_D = \mathrm{qlift}([Q]_d, D)$ as described in Proposition 3.9.3 using sequential Hensel-lifting ($E = 1$).

**input** : $[A]_D = [A_{[0]}, \ldots, A_{[D-1]}]$, where $A_{[d]} \in \mathbb{R}^{N \times N}$ symmetric, $d = 0, \ldots, D-1$
**output**: $[\Lambda]_D = [\Lambda_{[0]}, \ldots, \Lambda_{[D-1]}]$, where $\Lambda_{[0]} \in \mathbb{R}^{N \times N}$ diagonal and $\Lambda_{[d]} \in \mathbb{R}^{N \times N}$ block diagonal $d = 1, \ldots, D-1$.
**output**: $[Q]_D = [Q_{[0]}, \ldots, Q_{[D-1]}]$ orthogonal, where $Q_{[d]} \in \mathbb{R}^{N \times N}$
**output**: $b \in \mathbb{N}^{N_b+1}$, array of integers defining the blocks. The integer $N_b$ is the number of blocks. Each block has the size of the multiplicity of an eigenvalue $\lambda_{n_b}$ of $\Lambda_{[0]}$ s.t. for $s = b_{n_b} : b_{n_b+1} - 1$ one has $(Q_{[0];:,s})^T A_{[0]} Q_{[0];:,s} = \lambda_{n_b} \mathbf{1}$.

$\Lambda_{[0]}, Q_{[0]} = \mathrm{eigh}(A_{[0]})$

compute $b \in \mathbb{R}^{N_b+1}$
$E_{ij} = \Lambda_{[0];jj} - \Lambda_{[0];ii}$
$H = P_b \circ (1/E)$

**for** $d = 1$ **to** $D - 1$ **do**
  $\Delta F = \sum_{|\mathbf{i}|=d} Q_{[i_1]}^T A_{[i_2]} Q_{[i_3]}$
  $S = -\frac{1}{2} \sum_{k=1}^{d-1} Q_{[d-k]}^T Q_{[k]}$
  $K = \Delta F + Q_{[0]}^T A_{[d]} Q_{[0]} + S\Lambda_{[0]} + \Lambda_{[0]} S$
  $Q_{[d]} = Q_{[0]}(S + H \circ K)$
  $\Lambda_{[d]} = \bar{P}_b \circ K$
**end**

**Algorithm 3.9.2:** This algorithm computes $[\Lambda]_D, [Q]_D, b = \mathrm{eigh}_1([A]_D)$ as specified by 3.9.2 using sequential Hensel-lifting ($E = 1$). I.e., the zeroth coefficient is diagonalized and the higher order coefficients are block diagonalized. The symbol $i \in \mathbb{N}_0^3$ denotes a multiindex, i.e., the summation $\sum_{|i|=d}$ goes over all possible $i$ such that $|i| \equiv \sum_{k=1}^3 i_k = d$.

In this example it is the goal to evaluate the Taylor series expansion of $f(x(t))$ about $t = 0$, where $x(t) = \binom{0}{1} t$ for $t \in [-1, 1]$ by using by using Algorithm 3.9.3. I.e., let $[x]_2 = [0, \binom{0}{1}]$ be given from which it directly follows that

$$[A]_2 = \left[ \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \right] \equiv \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} T \, .$$

To compute the Taylor polynomials of eigenvalues $[\lambda]_2 \in \mathbb{R}[T]/(T^D)^2$ one proceeds as follows:

1. Set $[\Lambda^0]_2 = [A]_2$, $[Q^0]_2 = \mathbf{1}$ and the blocksize delimiter $b^0 = [1, 3]$.

**input** : $[A]_D = [A_{[0]}, \ldots, A_{[D-1]}]$ symmetric with $A_{[d]} \in \mathbb{R}^{N\times N}$
**output**: $[\Lambda]_D = [\Lambda_{[0]}, \ldots, \Lambda_{[D-1]}]$, where $\Lambda_{[d]} \in \mathbb{R}^{N\times N}$ diagonal for $d = 0, \ldots, D-1$.
**output**: $[Q]_D = [Q_{[0]}, \ldots, Q_{[D-1]}]$ orthogonal, where $Q_{[d]} \in \mathbb{R}^{N\times N}$
$[\Lambda^0]_D = [A]_D$, $[Q^0]_D = \mathbf{1}$ and $b^0 = [1, N+1]$
**for** $d = 0$ **to** $D-1$ **do**
  **for** $n_b = 1$ **to** $N_b^d$ **do**
    $s = b^d_{n_b} : b^d_{n_b+1} - 1$  (slice index)
    $[\hat\Lambda_{s,s}]_{D-d}, [\hat Q_{s,s}]_{D-d}, b^{d+1} = \mathrm{eigh}_1([\Lambda^d_{s,s}]_{d:})$
    $[Q_{s,s}]_D = \mathrm{qlift}([\hat Q_{s,s}]_{D-d}, D)$
  **end**
  $[\Lambda^{d+1}]_D = [\Lambda^d]_d + [\hat\Lambda]_{D-d} T^d$
  $[Q^{d+1}]_D = [Q^d]_D [Q]_D$
**end**

**Algorithm 3.9.3:** This algorithm computes $[\Lambda]_D, [Q]_D = \mathrm{eigh}([A]_D)$ as described in Theorem 3.9.1. The algorithm uses internally Algorithm 3.9.2 and 3.9.1. The loop $d = 0, \ldots, D-1$ increases the level $d \mapsto d+1$. At each level the matrix is already block-diagonal and diagonal up to degree $d$.

2. Then a call to $\mathrm{eigh}_1$ returns

$$\begin{aligned}[\hat\Lambda]_2 &= \left[\begin{pmatrix}0&0\\0&0\end{pmatrix}, \begin{pmatrix}0&1\\1&0\end{pmatrix}\right]\\ [\hat Q]_2 &= \left[\begin{pmatrix}1&0\\0&1\end{pmatrix}, \begin{pmatrix}0&0\\0&0\end{pmatrix}\right]\\ b^1 &= [1,3]\ .\end{aligned}$$

   I.e., $\mathrm{eigh}_1$ doesn't really do any work since the input already satisfies the relaxed equations of level 1 and order 2. The slice index $s = 1:2$ has been dropped because it selects here the complete matrix. For other examples one would have to work with slice indices.

3. In the following iteration (when $d = 1$) the higher-order coefficients get block-diagonalized. In this example (order $D = 2$) there is just one more coefficient, namely

$$[\Lambda^1]_{1:} = \left[\begin{pmatrix}0&1\\1&0\end{pmatrix}\right]$$

   (reminder: $[A_{[0]}, A_{[1]}, A_{[2]}]_{1:3} = [A_{[1]}, A_{[2]}]_2$, i.e. all coefficients without the 0-th) is the input for $\mathrm{eigh}_1$ and one obtains

$$\begin{aligned}[\hat\Lambda]_1 &= \left[\begin{pmatrix}-1&0\\0&1\end{pmatrix}\right]\\ [\hat Q]_1 &= \frac{1}{\sqrt{2}}\left[\begin{pmatrix}-1&1\\1&1\end{pmatrix}\right]\\ b^1 &= [1,2,3]\ .\end{aligned}$$

These matrices are used to compute

$$[\Lambda^2]_2 = [\Lambda^1]_1 + [\hat{\Lambda}]_1 T = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} T$$
$$[Q^2]_2 = \mathbf{I}\hat{Q}_{[0]} + \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} T \,,$$

where the last equation holds since $\mathrm{qlift}([\hat{Q}]_1, 2) = [\hat{Q}_{[0]}, \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}]$. This is already the desired result since $[Q]_2 \equiv [Q^2]_2$ and $[\Lambda]_2 \equiv [\Lambda^2]_2$.

In Figure 3.4 both eigenvalues are plotted along the path $x(t) = \binom{0}{1} t$ for $t \in [-1, 1]$ and additionally the first order Taylor series expansion evaluated at $[x]_2 = [0, \binom{0}{1}]$.

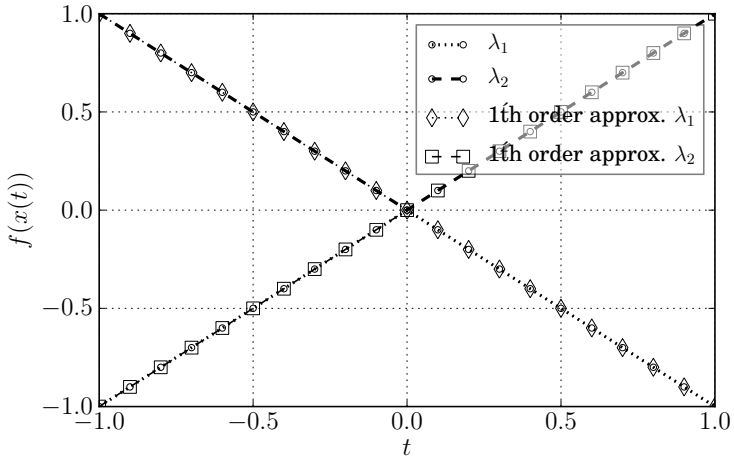



Figure 3.4.: Function and Taylor series expansion as explained in Example 3.9.1. One can see that the largest eigenvalue $\lambda_2$ as well as the other eigenvalue $\lambda_1$ do not depend smoothly on $t$ but have a kink in zero. Nonetheless it is possible to compute the directional derivative at $t = 0$.

*Remark.* Consider the task of computing the arithmetic mean of the $M$ largest eigenvalues of $A(x) \in \mathbb{R}^{N \times N}$

$$f(x) = \frac{1}{M} \sum_{m=1}^{M} \lambda_m(A(x)) = \frac{1}{M} \operatorname{tr}(\Lambda^0_{:M,:M}(x)) \,,$$

and the geometric mean

$$g(x) = \left( \prod_{m=1}^{M} \lambda_m(A(x)) \right)^{\frac{1}{M}} = \left( \det(\Lambda^0_{:M,:M}(x)) \right)^{\frac{1}{M}} \,,$$

where $\Lambda_{:M,:M}$ is associated to the $M$ largest eigenvalues such that the defining equations of level 0 are satisfied. For $y \geq 0,\ m = 1, \ldots, M$ one has the inequality

$$\frac{1}{M} \sum_{m=1}^{M} y_m \geq \left( \prod_{m=1}^{M} y_m \right)^{\frac{1}{M}}$$

and equality holds if and only if $y_1 = \cdots = y_M = 0$. Hence, one could in principle use this to enforce that eigenvalues coalesce. If $A(x)$ is indefinite one can add a multiple of the identity

matrix.

**Example 3.9.2.** Consider the following test example introduced by Andrew and Tan [1998]: They define a diagonal matrix

$$\Lambda(t) = \operatorname{diag}(x^2 - x + \frac{1}{2}, 4x^2 - 3x, \delta(-\frac{1}{2}x^3 + 2x^2 - \frac{3}{2}x + 1) + (x^3 + x^2 - 1), 3x - 1)$$

where $x \equiv x(t) := 1 + t$ and $\delta$ is some predefined constant. One can see that the coefficients are analytic functions in $t$. The corresponding Taylor coefficients are

$$\begin{aligned}
\Lambda_{[0]} &= \operatorname{diag}(1/2, 1, 1 + \delta, 2) \\
\Lambda_{[1]} &= \operatorname{diag}(1, 5, 5 + \delta, 3) \\
\Lambda_{[2]} &= \operatorname{diag}(2, 8, 8 + \delta, 0) \\
\Lambda_{[3]} &= \operatorname{diag}(0, 0, 6 - 3\delta, 0) \\
\Lambda_{[d]} &= \operatorname{diag}(0, 0, 0, 0), \quad \forall d \geq 4 \, .
\end{aligned}$$

Using the orthogonal matrix

$$Q(t) = \frac{1}{\sqrt{3}} \begin{pmatrix} \cos(x(t)) & 1 & \sin(x(t)) & -1 \\ -\sin(x(t)) & -1 & \cos(x(t)) & -1 \\ 1 & -\sin(x(t)) & 1 & \cos(x(t)) \\ -1 & \cos(x(t)) & 1 & \sin(x(t)) \end{pmatrix}$$

one can create a symmetric matrix

$$A(t) := Q(t)^T \Lambda(t) Q(t)$$

with the same eigenvalues as $\Lambda(t)$. Writing $A(t)$ as truncated univariate Taylor polynomial $[A]_D$ one can reconstruct the Taylor polynomial approximation of $\Lambda(t)$

$$[\tilde{\Lambda}]_D, [\tilde{Q}]_D = \operatorname{eigh}([A]_D) \, .$$

If one plots the absolute error between $[\Lambda]_D$ and $[\tilde{\Lambda}]_D$ for $D = 5$ one finds the results depicted in Figure 3.5 and Figure 3.6.

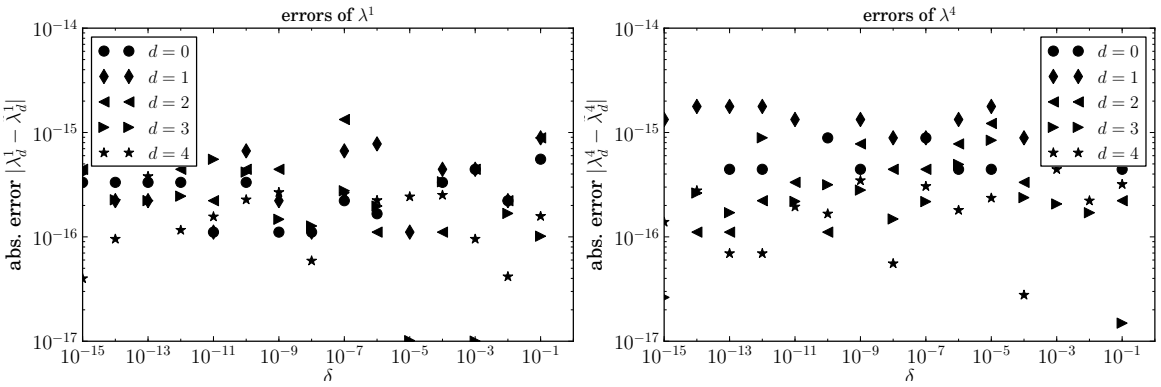



Figure 3.5.: The two eigenvalues $\lambda_1$ and $\lambda_4$ that are distinct. One can see that the accuracy is very good for all orders.

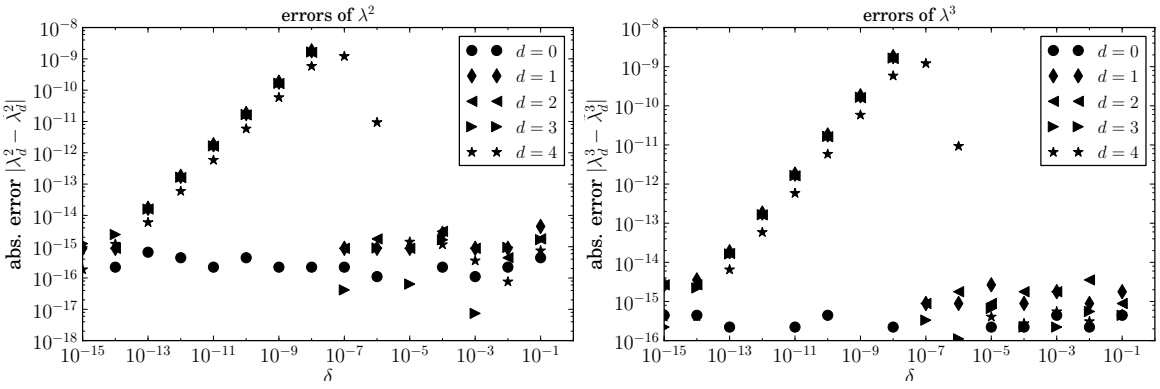



Figure 3.6.: The two eigenvalues $\lambda_2$ and $\lambda_3$ that are very close. The algorithm treats eigenvalues that satisfy $|\lambda_i - \lambda_j| < 10^{-7}$ as repeated eigenvalues.

### 3.9.2. Pullback

The eigenvalue decomposition is non-differentiable at points where eigenvalues are repeated and hence the defining equations do not define a *well behaved implicit mapping* as described by Christianson [1998]. However, the eigenvalue decomposition is typically used within a global context where the non-uniqueness and non-differentiability can be worked around. Here, only the pullback algorithm that is correct for unique eigenvalues is shown. The following proposition has also been described by Giles [2007].

**Proposition 3.9.4** (Pullback of the Symmetric Eigenvalue Decomposition with Distinct Eigenvalues:)**.** *Given $A, Q, \Lambda, \bar{Q}, \bar{\Lambda}$, where all eigenvalues are distinct, one can compute $\bar{A}$ by*

$$H_{ij} = (\lambda_j - \lambda_i)^{-1} \quad \text{if} \quad i \neq j, \quad 0 \quad \text{else} \tag{3.51}$$

$$\bar{A} = Q\left(\bar{\Lambda} + H \circ (Q^T \bar{Q})\right) Q^T \ . \tag{3.52}$$

*Proof.* It is desired to satisfy $\mathrm{tr}(\bar{A}^T \dot{A}) = \mathrm{tr}(\bar{\Lambda}^T \dot{\Lambda}) + \mathrm{tr}(\bar{Q}^T \dot{Q})$. In a first step, differentiate the implicit system

$$\begin{aligned} 0 =& Q^T A Q - \Lambda \\ 0 =& Q^T Q - \mathbf{I} \\ 0 =& (P_L + P_R) \circ \Lambda \end{aligned}$$

and obtain

$$\begin{aligned} \dot{\Lambda} =& Q^T \dot{A} Q + \dot{Q}^T A Q + Q^T A \dot{Q} \\ =& Q^T \dot{A} Q + \dot{Q}^T Q \Lambda + \Lambda Q^T \dot{Q} \\ 0 =& \dot{Q}^T Q + Q^T \dot{Q} \ . \end{aligned}$$

A straight forward calculation shows:

$$\begin{aligned} \mathrm{tr}(\bar{\Lambda}^T \dot{\Lambda}) &= \mathrm{tr}(Q \bar{\Lambda} Q^T \dot{A}) + \mathrm{tr}(\Lambda \bar{\Lambda} \dot{Q}^T Q) + \mathrm{tr}(\bar{\Lambda} \Lambda Q^T \dot{Q}) \\ &= \mathrm{tr}(Q \bar{\Lambda} Q^T \dot{A}) \ , \\ \mathrm{tr}(\bar{Q}^T \dot{Q}) &= \mathrm{tr}(\bar{Q}^T Q (H \circ (Q^T \dot{A} Q))) \\ &= \mathrm{tr}(Q (H^T \circ (\bar{Q}^T Q)) Q^T \dot{A}) \ , \\ \mathrm{tr}(\bar{A}^T \dot{A}) &= \mathrm{tr}\left((Q(\bar{\Lambda} + H^T \circ (\bar{Q}^T Q)) Q^T) \dot{A}\right) \end{aligned}$$

where it has been used that

$$\begin{aligned}\dot{\Lambda} =& Q^T\dot{A}Q - (Q^T\dot{Q})\Lambda + \Lambda Q^T\dot{Q}\\ =& Q^T\dot{A}Q - K \circ (Q^T\dot{Q})\\ \Longrightarrow\ Q^T\dot{Q} =& H \circ (Q^T\dot{A}Q - \dot{\Lambda})\\ =& H \circ (Q^T\dot{A}Q)\ .\end{aligned}$$

The matrix $K \in \mathbb{R}^{N\times N}$ is defined by $K_{ij} := \Lambda_{jj} - \Lambda_{ii}$ and $H_{ij} = (K_{ij})^{-1}$ for $i \neq j$ and $H_{ij} = 0$ otherwise and used the property $X\Lambda - \Lambda X = K \circ X$ for all $X \in \mathbb{R}^{N\times N}$ and diagonal $\Lambda \in \mathbb{R}^{N\times N}$. □

## 3.10. Numerical Tests

All algorithms of this chapter are implemented in the Python AD tool AlgoPy [Walter, 2009]. The purpose of this section is to perform a couple of simple numerical tests to illustrate the numerical properties of the derived algorithms.

### 3.10.1. Determinant via Matrix Factorizations

The determinant of a matrix $A$ is defined as the sum of products

$$\det(A) = \sum_{\sigma\in S_N} \operatorname{sgn}(\sigma) \prod_{i=1}^{N} A_{i,\sigma_i}\ ,$$

where $S_N$ is the symmetric group. The function $\det(A)$ is analytical in each element of $A$ since the evaluation involves only (finitely many) additions and multiplications. One can evaluate the determinant in many other ways, e.g., using an LU, SVD or QR decomposition. If $A$ is symmetric, it is also possible to use a Cholesky or symmetric eigenvalue decomposition.

The test is set up as follows: At first, a matrix $A(x) = Q(x)\Lambda(x)Q(x)^T$ is defined, where

$$\begin{aligned}\Lambda(x) =& \begin{pmatrix}\sin(x^2)+1 & & & \\ & \log(x^2+2) & & \\ & & 1. & \\ & & & \cos(5x)+1\end{pmatrix}\\ Q(x) =& \frac{1}{\sqrt{3}}\begin{pmatrix}\cos(x) & 1 & \sin(x) & -1\\ -\sin(x) & -1 & \cos(x) & -1\\ 1 & -\sin(x) & 1 & \cos(x)\\ -1 & \cos(x) & 1 & \sin(x)\end{pmatrix}\ .\end{aligned}$$

Since $Q(x)$ is orthogonal for all $x$, it follows that

$$\det(Q(x)\Lambda(x)Q(x)^T) = \det(\Lambda(x)) = \prod_{n=1}^{N}\Lambda_{nn}(x)\ .$$

Now, let $x \equiv x(T) = x + T$ and assume that the Taylor series expansion of

$$\sum_{d=0}^{3}\tilde{y}_{[d]}T^d := \prod_{n=1}^{N}\Lambda_{nn}(x(T)) + \mathcal{O}(T^4)$$

can be evaluated with a relative error close to the machine precision. Then, this result is

compared to the Taylor series expansion

$$\sum_{d=0}^{3} y_{[d]} T^d := \det(A(x(T))) + \mathcal{O}(T^4),$$

where one of the matrix factorization is used internally. Python code for several algorithmic variants can be found in Listing 3.2. For the test, the first $D = 4$ coefficients $[y]_4 = \sum_{d=0}^{3} y_{[d]} T^d$ are evaluated at $x(T) = x + T$, where $x \in \{10^{-8}, \ldots, 2\}$. The diagonal elements of $\Lambda(x)$ are plotted in Figure 3.7 and the relative and absolute errors $\left|\frac{\tilde{y}_{[d]} - y_{[d]}}{\tilde{y}_{[d]}}\right|$ and $\tilde{y}_{[d]} - y_{[d]}$ are shown in Figure 3.8. One can see that when the condition number of $A(x)$ is large (as it happens near 0.6 and 1.8) then also the error gets relatively large. One should note that also the dot products in $Q(x)\Lambda(x)Q(x)^T$ introduce an error. I.e., even if the decompositions were precise, there would be an error. A detailed error analysis would be interesting but is beyond the scope of this study. Otherwise, the relative errors are close to the machine precision.

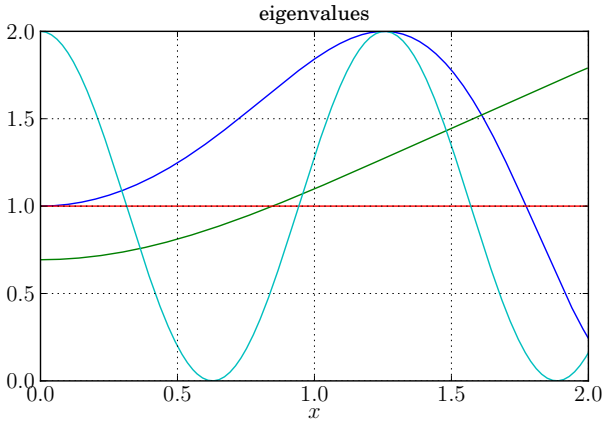



Figure 3.7.: The numerical values $\mathrm{diag}(\Lambda(x))$.

```
import algopy; import numpy

def det_cholesky(A):
    """ determinant of a symm. pos. def. matrix
    using the Cholesky decomposition
    """
    L = algopy.cholesky(A)
    return algopy.prod(algopy.diag(L))**2

def det_eigh(A):
    """ determinant of a symm. pos. def. matrix
    using the symm. eig. val. decomposition
    """
    l,Q = algopy.eigh(A)
    return algopy.prod(l)

def det_qr(A):
    """ determinant of a matrix
    using the QR decomposition
    """
    Q,R = algopy.qr(A)
    return algopy.prod(algopy.diag(R))

def det_svd(A):
    """ determinant of a full rank matrix
    using the SVD decomposition
    """
    N = A.shape[0]
    tmp = algopy.zeros((2*N, 2*N), dtype=A)
```

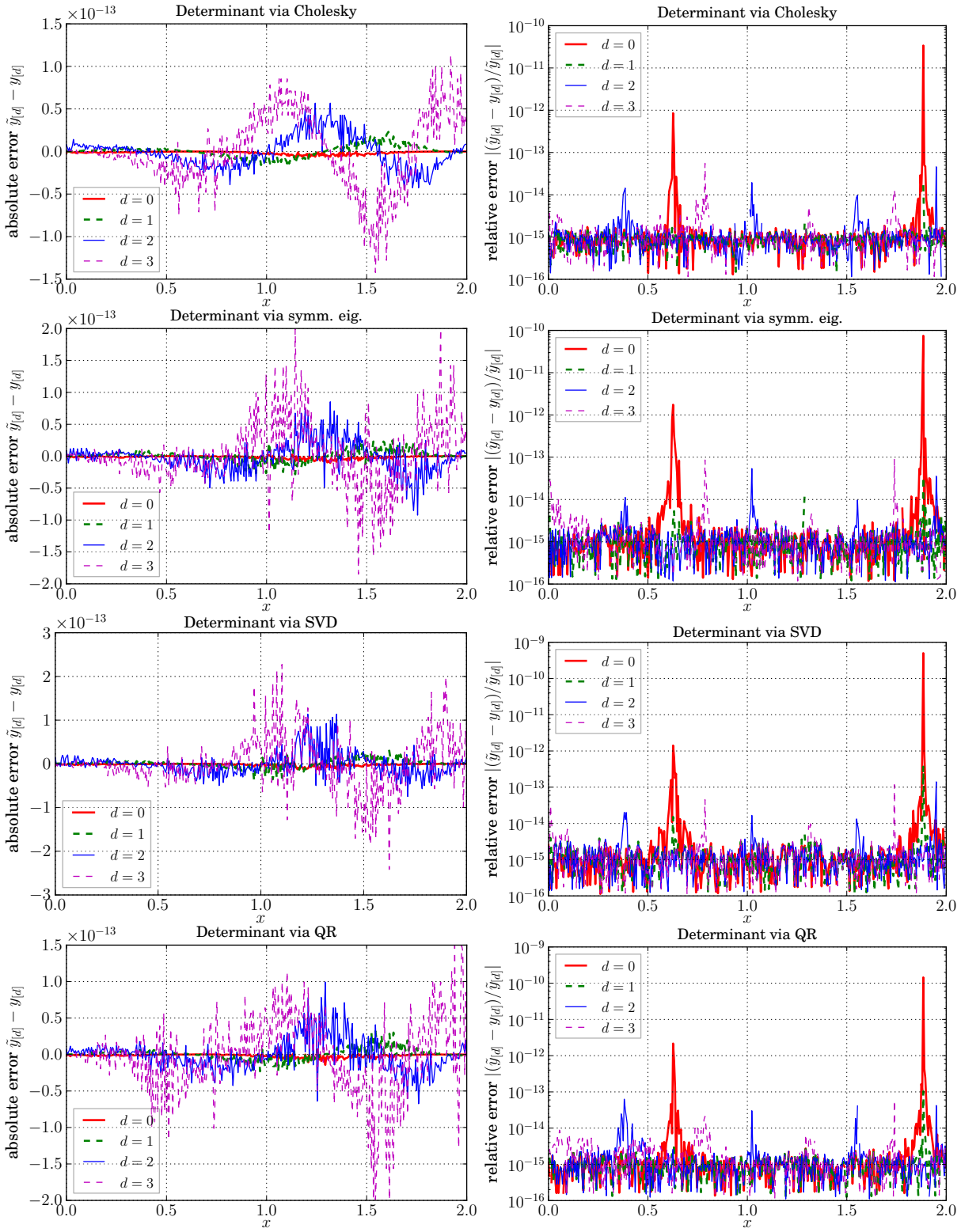



Figure 3.8.: The absolute error between the coefficients of the "true" Taylor polynomial $[\tilde{y}]_4$ and the numerically computed $[y]_4$. "sym. eig." is the symmetric eigenvalue decomposition. See Chapter 3.10.1 for the full discussion.

```
30      tmp[:N,N:] = A
        tmp[N:,:N] = A.T
        l,Q = algopy.eigh(tmp)
        return algopy.prod(l[:N])
```

```
def eval_Q(x):
    sin = algopy.sin; cos = algopy.cos
    Q = algopy.zeros((4,4),dtype=x)
    Q[0,0] = cos(x);  Q[0,1] = 1.;      Q[0,2] = sin(x); Q[0,3] = -1.
    Q[1,0] = -sin(x); Q[1,1] = -1.;     Q[1,2] = cos(x); Q[1,3] = -1.
    Q[2,0] = 1.;      Q[2,1] = -sin(x); Q[2,2] = 1;      Q[2,3] = cos(x)
    Q[3,0] = -1.;     Q[3,1] =  cos(x); Q[3,2] = 1;      Q[3,3] = sin(x)
    Q /= 3**0.5
    return Q

def eval_lam(x):
    lam = algopy.zeros(4, dtype = x)
    lam[0] = algopy.sin(x**2)+1
    lam[1] = algopy.log(x**2+2)
    lam[2] = 1.
    lam[3] = algopy.cos(5*x)+1
    return lam
```

Listing 3.2: Several algorithms to compute the determinant. See Chapter 3.10.1 for the discussion.

### 3.10.2. Covariance Matrix

The numerical evaluation of

$$\begin{aligned} f : \mathbb{R}^{M\times N} \times \mathbb{R}^{K\times N} &\to \mathbb{R} \\ (J_1, J_2) &\mapsto y = \operatorname{tr}(C) \end{aligned}$$

$$\text{where} \quad C = C(J_1, J_2) = (\mathbf{1}, 0) \begin{pmatrix} J_1^T J_1 & J_2^T \\ J_2 & 0 \end{pmatrix}^{-1} \begin{pmatrix} \mathbf{1} \\ 0 \end{pmatrix}$$

plays a significant role in Chapter 6. As discussed in the introduction of Chapter 3, it is possible to compute $C$ equivalently by using the formula

$$C = Q_2^T \left( Q_2 J_1^T J_1 Q_2^T \right)^{-1} Q_2 \,,$$

where the matrix $Q_2$ spans the nullspace of $J_2$ and can be computed by a QR decomposition $J_2^T = (Q_1^T, Q_2^T)(L, 0)^T$. In Listing 3.3 one can find a Python code where both alternatives to compute $C$ are implemented as the functions `eval_C` and `eval_C_qr`. As one can see in the code, the matrices $J_1 \in \mathbb{R}^{4\times 2}$ and $J_2 \in \mathbb{R}^{1\times 2}$ are defined to be matrix valued functions of $x \in \mathbb{R}^2$ and are implemented in `eval_J1` and `eval_J2`.

In the first test, the univariate Taylor polynomial $[x]_6 = \binom{1}{1} + \binom{3}{1} T$ is propagated through `eval_J1` and `eval_J2` to obtain numerical values for $[J_1]_6$ and $[J_2]_6$. Then, these matrix polynomials are propagated through `eval_C` and `eval_C_qr` and one obtains the numerical values $[C_1]_6$ and $[C_2]_6$. Finally, the relative difference

$$\frac{|C_{1,[d];ij} - C_{2,[d];ij}|}{\min(|C_{1,[d];ij}|, |C_{2,[d];ij}|)}$$

is computed for all $i, j = 1, \ldots, N$. In Figure 3.9 the largest of these relative differences is shown for $d = 0, \ldots, 5$.

As a second test, the gradient

$$\nabla_x f(C(J_1(x), J_2(x)))$$

and the Hessian

$$\nabla_x^2 f(C(J_1(x), J_2(x)))$$

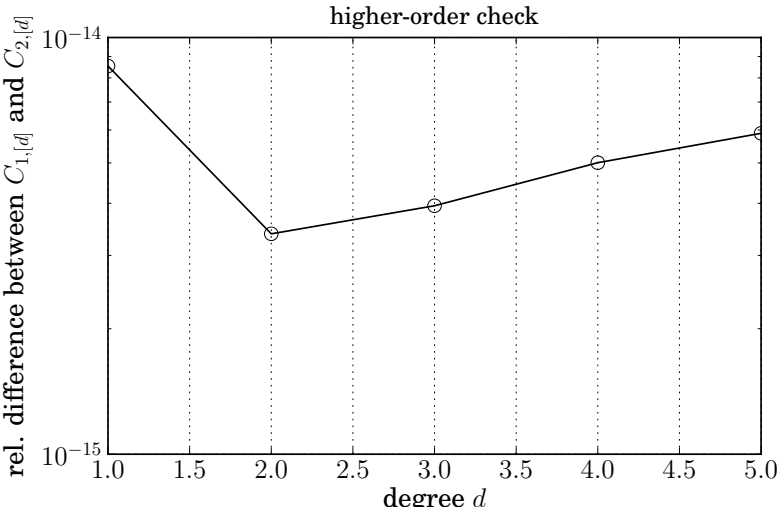



Figure 3.9.: The relative difference in the Taylor coefficients computed using `eval_C` and `eval_C_qr` is close to the machine precision.

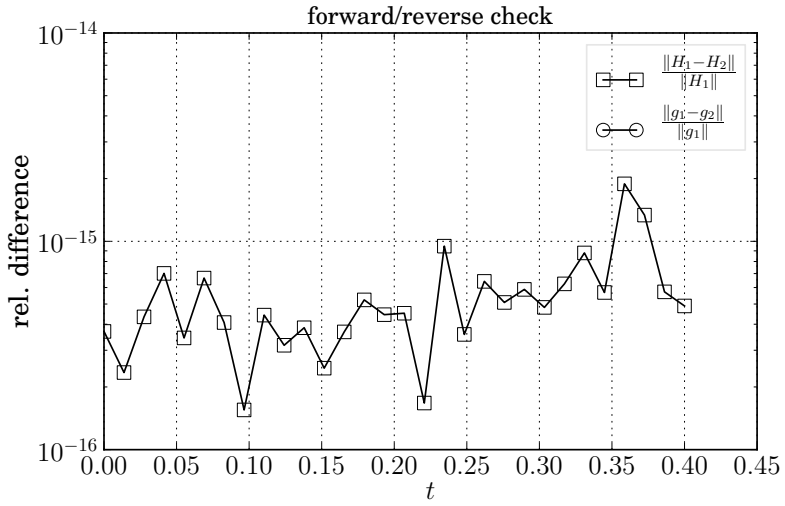



Figure 3.10.: The relative difference for gradient and Hessian evaluation. The gradients $g_1(x(t))$ and $g_2(x(t))$ are evaluated using using the forward resp. reverse mode of AD. The Hessians $H_1(x(t))$ and $H_2(x(t))$ are evaluated using a combined forward/reverse accumulation resp. univariate Taylor polynomial arithmetic in combination with a polarization identity. The relative error of the gradient is smaller than the machine precision and therefore not shown in this plot.

shall be computed using the algorithm `eval_C_qr`. The gradient can be computed either in the forward mode or in the reverse mode. One possibility to compute the Hessian is to use second order univariate Taylor polynomial arithmetic followed by the application of the polarization identity 2.21. Another is to use a combined forward/reverse mode. Both results should, naturally, coincide. Thirty sample points are generated by the function `eval_x` where $t \in [0, 2/5]$ is used as input. In Figure 3.10 the observed values are shown.

```
import numpy; import algopy
from algopy import UTPM, zeros, CGraph, Function, dot, eigh, qr
from algopy import qr_full, inv, solve, trace, sin, cos, log, exp
4
```

```
def eval_x(t):
    return numpy.array([1.,1.]) + t*numpy.array([3.,1.])

def eval_J1(x):
    retval = zeros((4,2), dtype=x)
    retval[0,0] = sin(x[0])*x[1]
    retval[0,1] = cos(x[1])
    retval[1,0] = exp(x[1])
    retval[1,1] = x[0]*x[1]
    retval[2,0] = x[0]*log(x[1])
    retval[2,1] = log(1 + exp(cos(x[0])))
    retval[3,0] = x[0] + x[1]
    retval[3,1] = x[0]*(x[1] + cos(x[0]))
    return retval

def eval_J2(x):
    retval = zeros((1,2), dtype=x)
    retval[0,0] = x[0] * log( x[1] + 3*sin(x[0]*x[1]))
    retval[0,1] = x[1] * exp(sin(x[0]) + cos(x[0]*x[1]))
    return retval

def eval_C(J1, J2):
    M,N = J1.shape
    K,N = J2.shape
    retval = zeros((N+K,N+K), dtype=J1)
    retval[:N,:N] = dot(J1.T, J1)
    retval[N:,:N] = J2
    retval[:N, N:] = J2.T
    return inv(retval)[:N,:N]

def eval_C_qr(J1, J2):
    M,N = J1.shape
    K,N = J2.shape
    Q,R = qr_full(J2.T)
    Q2a = Q[:,K:]
    Q2  = Q2a.T
    Q,R = qr(dot(J1, Q2a))
    V = solve(R.T, Q2)
    return dot(V.T,V)

def eval_f(C):
    return trace(C)
```

Listing 3.3: This listing shows Python code to evaluate derivatives of the function $f$ via two different algorithms. See Chapter 3.10.2 for the discussion.

## 3.11. Runtime Comparison

Not only the correctness and the theoretical computational complexity of an algorithm are of interest, but also how fast the algorithm can be executed on a computer. As discussed above, there are two possibilities how numerical linear algebra functions can be evaluated in the forward and reverse mode of AD: Either by considering the algorithm to be a sequence of basic elementary functions – univariate Taylor polynomial arithmetic of scalars, UTPS –, or by employing the matrix calculus approach – univariate Taylor polynomial arithmetic of matrices, UTPM –.

**Theoretical Considerations** To get an idea how the two possibilities compare in terms of actual execution speed, it is instructive to get a theoretical idea what one can expect. For simplicity, the recurrence defined in Algorithm 3.5.1 is considered. It yields a computational cost of

$$\mathrm{ops}(-X^{-1}) + \frac{(D-1)(D-2)}{2}\mathrm{ops}(+) + \frac{(D+2)(D-1)}{2}\mathrm{ops}(\mathrm{dot}) \ .$$

Assuming that the matrix addition is an $\mathcal{O}(N^2)$ and both the matrix multiplication and inversion (dot, inv) are $\mathcal{O}(N^3)$ operations, one obtains a computational cost that scales with $\mathcal{O}(D^2N^3)$. On the other hand, evaluating the matrix inversion in UTPS arithmetic requires

$$\mathrm{ops}(\cdot, X^{-1})\left(\frac{(D-1)D}{2}\mathrm{ops}(x+y) + \frac{(D+1)D}{2}\mathrm{ops}(xy)\right) + \mathrm{ops}(+, X^{-1})\left(D\mathrm{ops}(x+y)\right)$$

operations in total. The quantities $\mathrm{ops}(\cdot, X^{-1})$ and $\mathrm{ops}(+, X^{-1})$ are the number of multiplications respectively additions in the matrix inversion. In the leading powers it is also $\mathcal{O}(N^3D^2)$. Hence, one can expect UTPM and UTPS arithmetic to show the same scaling law in a runtime comparison.

Nonetheless, there are reasons why one can expect a relatively large difference anyway. Firstly, simply counting the operations is inadequate since there is no one-to-one correspondence between the mathematical formulation and the sequence of instructions that are performed on the hardware. In particular, the mathematical formulation has no notion of memory movements. As practitioner, one is typically not interested in such details and therefore tries to use optimized implementations such as provided by ATLAS (c.f. [Whaley et al., 2001]). Looking at Algorithm 3.5.1 one can see that one can compute all coefficients $Y_{[d]}$ by calling the ATLAS routines *clapack_dgetrf, clapack_dgetri* and *cblas_dgemm*. I.e., using UTPM arithmetic it is possible to employ existing high-performance implementations. The alternative, i.e., augmenting such optimized implementations for UTPS arithmetic, is likely to destroy the cache efficiency since a UTPS $[x]_D \in \mathbb{R}[T]/(T^D)$ requires $D$ times more memory than a scalar $x \in \mathbb{R}$.

The second reason concerns only the reverse mode of AD. Functions such as $\mathrm{dot}(\cdot,\cdot)$ and $\mathrm{inv}(\cdot)$ scale with $\mathcal{O}(N^3)$ and therefore $\mathcal{O}(N^3)$ intermediate values need to be available during a reverse sweep. Typically, AD tools simply write the intermediate values into memory during a forward evaluation and retrieve the intermediate values during the reverse sweep. That means that the naive UTPS approach requires memory of order $\mathcal{O}(D^2N^3)$. This has to be compared to the UTPM approach which requires to store only $\mathcal{O}(D^2N^2)$ intermediate values. Since modern CPUs are much faster than the memory, an $\mathcal{O}(D^2N^3)$ memory requirement has also a negative effect on the runtime performance. Note that not all is lost for the UTPS approach as it is, for certain algorithms, possible to recompute intermediate values as reported by Korelc [2010] for the LU decomposition or by Smith [1995] for the Cholesky decomposition. More generally, also checkpointing can be used reduce the memory requirement, see for instance [Griewank, 2003].

**Experiment Description** To compare the performance of UTPM and UTPS arithmetic, two easy but sufficiently complex examples are considered. In the first test problem, the gradient

$\nabla f(X) \in \mathbb{R}^{N \times N}$ of the function

$$X \mapsto f(X) = \mathrm{tr}(X^{-1}) \; . \tag{3.53}$$

is computed in the reverse mode of AD. Since the runtimes of the gradient evaluation depend on the underlying function, also the runtimes of the normal function evaluation are measured. To avoid problems due to pivoting, as it is mandatory in an LU decomposition, a matrix inversion algorithm based on a QR decomposition, employing Givens rotations, is used. Since ADOL-C requires a generic C++ code but Tapenade C code, the same algorithm is provided in both languages. ADOL-C requires the sequence of operations to be *traced* because all further operations with ADOL-C are performed on an internal representation of the function (see Chapter 2.1). The time for the tracing, as well as the time for a function evaluation (which is based on the internal representation), are measured. For comparison, also timings of an implementation using ATLAS are shown. The results are depicted and interpreted in Figure 3.11. More precisely, the algorithm first computes $Q, R = \mathrm{qr}(A)$ and then solves $RB = Q^T$, where $B = A^{-1}$. The implementation of the algorithm was done in C and C++. Hence, ADOL-C can be used to differentiate the C++ code and Tapenade to differentiate the C code. Also, the speed of our implementation of the matrix inversion is compared to the combined call of the LAPACK functions `dgetrf` and `dgetri`. This approach uses an LU decomposition with partial pivoting. The experimental results are shown in Figure 3.11.

The second test problem is the evaluation of the matrix product $\mathrm{dot} : \mathbb{R}^{N \times N} \times \mathbb{R}^{N \times N} \to \mathbb{R}^{N \times N}$ in UTP arithmetic. Since Tapenade does not offer the possibility of UTP arithmetic with $D > 2$, a dot function has been implemented where the elementary operations were replaced manually with their corresponding univariate Taylor polynomial instructions. As building blocks algorithms from Taylorpoly [Walter, 2010] are used. Furthermore, Taylorpoly's UTPM implementation of the dot function is utilized. The results are shown in Fig. 3.12.

The experiments have been performed on a Dell Latidude D530 with an Intel(R) Core(TM)2 Duo CPU T7300 @2.00GHz with 2048628 kB physical memory on Linux 2.6.32-24-generic. All sources have been compiled with gcc 4.4.3 using the optimization flag -O3. The source code for the tests is available at [Walter].

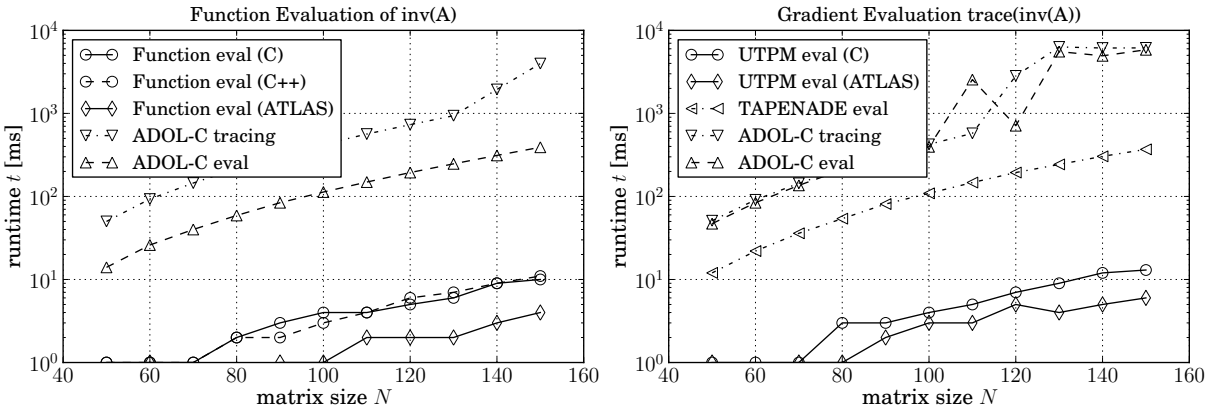



Figure 3.11.: The runtimes for different implementations to compute the inverse of a matrix are shown in the left plot . One can see that the tracing is the most time-consuming operation. Also, the interpreted evaluation of the function using the trace is much slower than the evaluation of compiled functions. In the right plot, one can see the runtime of the gradient evaluation. ADOL-C as well as the compiled code generated by Tapenade are significantly slower than the UTPM arithmetic.

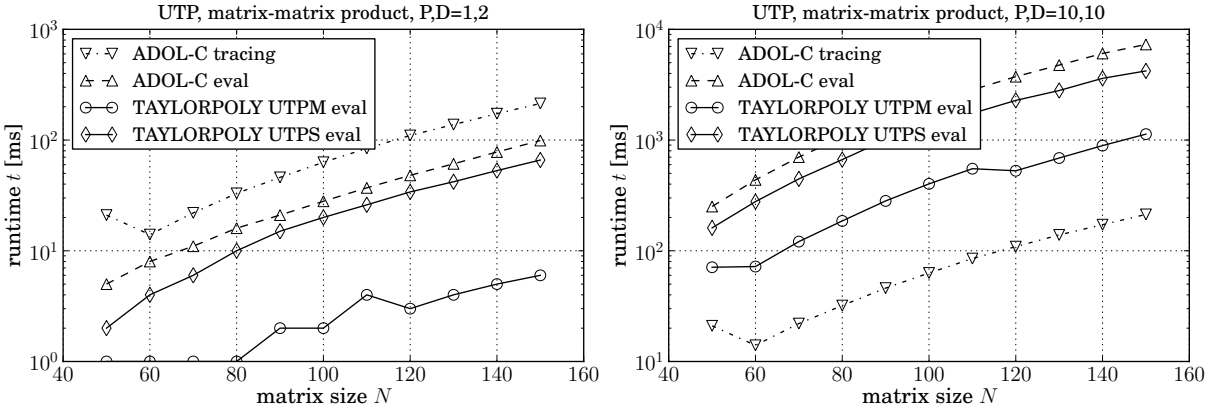



Figure 3.12.: In the left plot one can see that the UTPS arithmetic using Taylorpoly and ADOL-C is considerably slower than UTPM arithmetic, both for $D = 2$ and $D = 10$ as shown on the left respectively on the right. $P$ denotes the number of simultaneous directions as described in ADOL-C documentation Griewank et al. [1999] and is chosen as 10 to reduce the relative overhead of UTPS arithmetic.

# 4. Dynamical Systems

Systems which have a time-like dependence are called *dynamical systems*. In the following discussion, it is assumed that at each point of time the system takes a certain well-defined state

$$x(t; x_0, u, p) \in \mathbb{R}^{N_x} \ .$$

I.e., the state is regarded as function of *time* $t \in I := [t_0, t_e]$, *control function(s)* $u : I \to \mathbb{R}^{N_u}$, parameters $p \in \mathbb{R}^{N_p}$ and *initial value(s)* $x_0 \in \mathbb{R}^{N_x}$.

In practice, the state is generally not known in explicit form but only as solution to (partial) differential algebraic equations of the form

$$\begin{aligned} 0 =& M(t, x, u, p) \\ 0 =& x(t_0) - x_0 \ . \end{aligned} \tag{4.1}$$

Depending on $M$, such a system may not have a solution or it may not be unique. When it is unique one can ask how the system reacts to changes in $u, p$ or $x_0$. E.g., whether $x(t; x_0, u(t), p)$ is differentiable in $p$.

The chapter is structured as follows: At first, semi-implicit quasi-linear differential algebraic equations (DAEs) of index one are introduced in Chapter 4.1. In the following Chapter 4.2, it is described that the solution of index one DAEs can, under certain conditions, be reformulated as an initial value problems, i.e., the state can be regarded as a "well-behaved" function $x(t; x_0, u, p)$ of $x_0, u$ and $p$. For the direct approach, it is necessary to parametrize the control functions $u(t)$ by a finite dimensional control vector $q \in \mathbb{R}^{N_q}$. Several parametrizations of practical relevance are described in Chapter 4.3. Finally, a brief overview of the possible approaches to solve optimization problems with dynamical system constraints are given in Chapter 4.4.

## 4.1. Semi-Implicit Differential Algebraic Equations of Index One

Dynamical systems in chemical engineering can often be described by differential algebraic equations (DAEs). This is a consequence of the modeling process where reactions are of dynamical nature but have to satisfy conservation laws. Some reactions occur at completely different timescales and thus it is possible to consider certain reaction paths as instantaneous. In the most general form a DAE takes a fully-implicit nonlinear form

$$0 = F(t, \dot{x}, x, u, p) \ ,$$

where $F$ is an an algebraic function. There are several types of special cases that regularly appear in practice:

- Problems of the form
$$\frac{\mathrm{d}}{\mathrm{d}t} g(t, y, p) = f(t, y, u, p) \ .$$
where the function $g$ does not depend on the control function $u$.

- Semi-implicit quasi-linear DAEs

$$\begin{aligned} A(t,y,z,p)\dot{y} =& f(t,y,z,u,p) \\ 0 =& g(t,y,z,u,p) \ , \end{aligned}$$

  where $x = (y, z)$.

- Semi-explicit DAEs, which are semi-implicit quasi-linear DAEs with $A(t, y, z) = \mathbf{I}$, i.e.,

$$\begin{aligned} \dot{y} =& f(t,y,z,u,p) \\ 0 =& g(t,y,z,u,p) \ . \end{aligned}$$

The different formulations can be transformed into each other. E.g., $\frac{\mathrm{d}}{\mathrm{d}t} g(t, y, p) = f(t, y, u, p)$ can be written as a semi-implicit ODE by partial differentiation

$$\frac{\mathrm{d}g}{\mathrm{d}t}(t,y,p) = \frac{\partial g}{\partial y}(t,y,p)\bigg|_{y=y(t;y_0,u,p)} \dot{y} + \frac{\partial g}{\partial t}(t,y,p)$$

or as semi-explicit DAE by defining $z := g(t, y, p)$:

$$\begin{aligned} \dot{z} =& f(t,y,u,p) \\ 0 =& z - g(t,y,p) \ . \end{aligned}$$

Though such reformulations are often possible it may not be a good idea to do so, granted that an alternative exists. Algorithms that tackle more general problems typically cannot exploit the additional structure and therefore dedicated algorithms are often much more efficient and reliable in practice.

In the following, the focus is on dynamical models described by *semi-implicit quasi-linear DAEs of index one*

$$\begin{aligned} A(t,y,z,p)\dot{y} =& f(t,y,z,u,p) \\ 0 =& g(t,y,z,u,p) \ . \end{aligned} \tag{4.2}$$

To be able to solve such DAEs efficiently, one requires additional conditions, similar to constraint qualifications in nonlinear programming. The *index one* condition states that

$$A(t,y,z,p) \in \mathbb{R}^{N_y \times N_y} \quad \text{and} \quad \partial_z g(t,y,z,u,p) \in \mathbb{R}^{N_z \times N_z} \tag{4.3}$$

are both non-singular on an open domain containing all $t, y, z, u, p$ of interest. According to the discussion by Körkel [2002], one can use the implicit function theorem (Proposition 2.2.8) to find a local solution $z$ of $0 = g(t, y, z, u, p)$, denoted $z = \phi(t, y, u, p)$. When $g$ is $d$ times continuously differentiable, it follows that $\phi \in C^d$. Hence, the DAE reduces to an implicit ODE

$$A(t,y,\phi(t,y,u,p),p)\dot{y} = f(t,y,\phi(t,y,u,p),u,p) \ .$$

A successive left multiplication of $A^{-1}$ yields

$$\begin{aligned} \dot{y} =& A(t,y,\phi(t,y,u,p),p)^{-1} f(t,y,\phi(t,y,u,p),u,p) \\ =:& \tilde{f}(t,y,u,p) \ . \end{aligned}$$

I.e., under the index one assumption one can reformulate the DAE to an ODE. This possibility allows one to transfer existence and uniqueness results from the theory of ODEs to index one DAEs. For a discussion of the numerical aspects of the solution with a BDF integrator see

Albersmeyer [2005], Albersmeyer and Bock [2008], Brenan et al. [1996].

In the following, in particular in Chapter 5 and Chapter 6, the following nomenclature and assumptions will be useful:

- the variable $t \in I := [t_0, t_e] \subseteq \mathbb{R}$ is typically, but not always, the time
- the *model functions* $A, f, g$ are assumed to be sufficiently smooth
- $y : I \to \mathbb{R}^{N_y}$, $t \mapsto y(t) \in \mathbb{R}^{N_y}$ is called the *differentiable state*
- $z : I \to \mathbb{R}^{N_z}$, $t \mapsto z(t) \in \mathbb{R}^{N_z}$ is called the *algebraic state*
- $x : I \to \mathbb{R}^{N_x}$, $t \mapsto x(t) = (y(t), z(t)) \in \mathbb{R}^{N_y} \times \mathbb{R}^{N_z}$ the *state*
- $u : I \to \mathbb{R}^{N_u}$, $t \mapsto u(t) \in \mathbb{R}^{N_u}$ is called the *control function* with $u_i$ elements of suitable function spaces. Control variables are treated as constant functions.
- $p \in \mathbb{R}^{N_p}$ are parameters given by nature (cannot be controlled).
- $0 = g(t, y, z, u, p)$ is called the *algebraic equation.*

## 4.2. Initial Value Problem Formulation, Uniqueness/Existence of Solutions

In general, there can exist infinitely many solutions $x = (y, z)$ of (4.2). Additional constraints such as boundary values are necessary to obtain a unique solution. Though there is also the possibility to work directly in a function space setting, the uniqueness and existence results shown below allow it to describe the dynamics as initial value problem

$$\begin{aligned} A(t, y, z, p)\dot{y} &= f(t, y, z, u, p) \\ 0 &= g(t, y, z, u, p) \\ y(t_0) &= y_0 \ , \end{aligned} \tag{4.4}$$

where the initial values $y_0$ potentially are functions of constant control functions and parameters, i.e., $y_0 = y_0(t_0, u, p)$. As discussed in the previous section, it is possible to reformulate the DAE to an ODEs of the form

$$\begin{aligned} \dot{y}(t) &= \tilde{f}(t, y(t)) \in \mathbb{R}^{N_y} \ , \\ y(t_0) &= y_0 \ . \end{aligned}$$

I.e., the algebraic constraints are resolved. Additionally the control function $u(t)$ and parameters $p$ are included in $\tilde{f}$. Hence one can use the standard theorems for existence and uniqueness of solutions to ODEs and infer the existence and uniqueness of index one DAEs. More explicitly, the index one DAE is written as initial value problem. For a more in-depth discussion see Körkel [2002] but also Brenan et al. [1996], Deuflhard and Bornemann [2002, 1987].

**Proposition 4.2.1** (Peano existence theorem)**.** *Let $f : D \subseteq \mathbb{R} \times \mathbb{R}^{N_y} \to \mathbb{R}^{N_y}$ be continuous with initial values $(t_0, y_0)$. Then there exists an open neighborhood $W := U \times V$ of $(t_0, y_0)$ and at least one $y : U \to V$ satisfying*

$$\dot{y}(t) = f(t, y(t)) \ , \ y(t_0) = y_0 \ .$$

*Furthermore, the local solution can be extended to the boundary of $D$.*

*Proof.* See [Deuflhard and Bornemann, 2002, Chapter 2] or Deuflhard and Bornemann [1987] and references therein. □

**Proposition 4.2.2** (Picard-Lindelöf theorem)**.** *Let $f \in C^0(D)$ on the open set $D = (D_1, D_2) \subseteq \mathbb{R} \times \mathbb{R}^{N_y}$, and locally Lipschitz continuous for $y \in D_2$. I.e.,*

$$\|f(t,y) - f(t,\tilde{y})\| \leq L\|y - \tilde{y}\|$$

*for all $(t,y) \in D$ and $(t,\tilde{y}) \in D$. Then there exists for every point $t_0, y_0$ exactly one solution of the initial value problem $\dot{y} = f(t,y), y(t_0) = y_0$.*

*Proof.* See [Deuflhard and Bornemann, 2002, Chapter 2] or Deuflhard and Bornemann [1987] and references therein. □

Additionally to the uniqueness and existence, one would like to know how the solution of parameter dependent initial value problem

$$\begin{aligned} \dot{y} &= \tilde{f}(t,y,p) \\ y(t_0) &= y_0(t_0,p) \end{aligned}$$

depends on the parameters $p$. The following result puts the differentiability of the solution into relation with the differentiability of the model function $f$ and initial value function $y_0$. Note that the dependence on $u$ is here included in the model function. One arrives at the important result that the solution $x(t) = (y(t), z(t))$ of (4.4) can be described as a function

$$\varphi(t; y_0, z_0, u, p)$$

called the *flow*, see for instance Hairer et al. [2002] or Phipps [2003], granted that $z_0$ is consistent. I.e., given initial values, control function and the parameters the flow returns the state $x(t)$ at time $t$. It is convenient to introduce the notation

$$x(t; x_0, u, p) := \varphi(t; x_0, u, p)\ . \tag{4.5}$$

The semi-colon ; is used to separate a parametric dependence from the variables.

**Proposition 4.2.3.** *Let $f : D \subseteq \mathbb{R} \times \mathbb{R}^{N_y} \times \mathbb{R}^{N_p} \to \mathbb{R}^{N_y}$, $f \in C^d(D)$, then the solution $y(t; y_0, p)$ of the initial value problem $\dot{y} = f(t,y,p), y(t_0) = y_0$ is $d$-times continuously differentiable in $p$.*

*Proof.* See [Deuflhard and Bornemann, 2002, Chapter 3]. □

## 4.3. Control Function Parametrization

For a numerical simulation it is necessary to describe the control functions by a finite dimensional vector. E.g., by restriction to the parameterizable control functions

$$u(t) = u(t; q)\ ,$$

where the *control vector* $q \in \mathbb{R}^{N_q}$ defines the parametrization. The reason for this parametrization is two-fold: A parametrization is necessary for the numerical solution and furthermore the actual control functions that can be realized in a laboratory are typically of parametrizable nature. Examples are the spaces of

1. piecewise constant
2. piecewise linear
3. piecewise continuous linear
4. piecewise cubic

5. piecewise continuous cubic

functions. The above control functions $u(t;q)$ are defined piecewise, i.e., the total time horizon $[t_0, t_e]$ is partitioned into non-intersecting *control intervals*. Each control interval is given a unique id $n_{\mathrm{ci}} = 1, \ldots, N_{\mathrm{ci}}$. The function

$$\begin{aligned} c_i : \mathbb{N} &\to \mathbb{N} \\ n_{ci} &\mapsto n_q = c_i(n_{ci}) \end{aligned} \tag{4.6}$$

is a mapping from the index of a control interval $n_{ci}$ to the index of the parametrization vector $q$ of that interval. The integrator has to know at which point of time a new control interval starts. The vector of *switching times* of $u(t)$ is denoted $t_{\mathrm{ust}} \in \mathbb{R}^{N_{\mathrm{ci}}+1}$.

**Piecewise Constant** The control function $u(t)$ is parametrized by a control vector $q \in \mathbb{R}^{N_q}$, where $N_u = N_q$. The control interval mapping is $c_i(n_{\mathrm{ci}}) = n_{\mathrm{ci}}$. For $t \in [t_0, t_1]$ in control interval $n_{\mathrm{ci}}$ one has

$$u(t) = q_{n_{\mathrm{ci}}} \, . \tag{4.7}$$

**Piecewise Linear** The control function $u(t)$ is parametrized by a control vector $q \in \mathbb{R}^{N_q}$, where $N_q = 2N_{ci}$. The control interval mapping is $c_i(n_{ci}) = 2n_{ci} - 1$. For $t \in [t_0, t_1]$ in control interval $n_{ci}$ one has

$$u(t) = \frac{t_1 - t}{t_1 - t_0} q_{c_i(n_{ci})} + \frac{t - t_0}{t_1 - t_0} q_{(c_i(n_{ci})+1)} \, . \tag{4.8}$$

**Piecewise Linear Continuous** The control function $u(t)$ is parametrized by a control vector $q \in \mathbb{R}^{N_q}$, where $N_q = N_{ci} + 1$. The control interval mapping is $c_i(n_{ci}) = n_{ci}$. For $t \in [t_0, t_1]$ in control interval $n_{ci}$ one has

$$u(t) = \frac{t_1 - t}{t_1 - t_0} q_{c_i(n_{ci})} + \frac{t - t_0}{t_1 - t_0} q_{(c_i(n_{ci})+1)} \, . \tag{4.9}$$

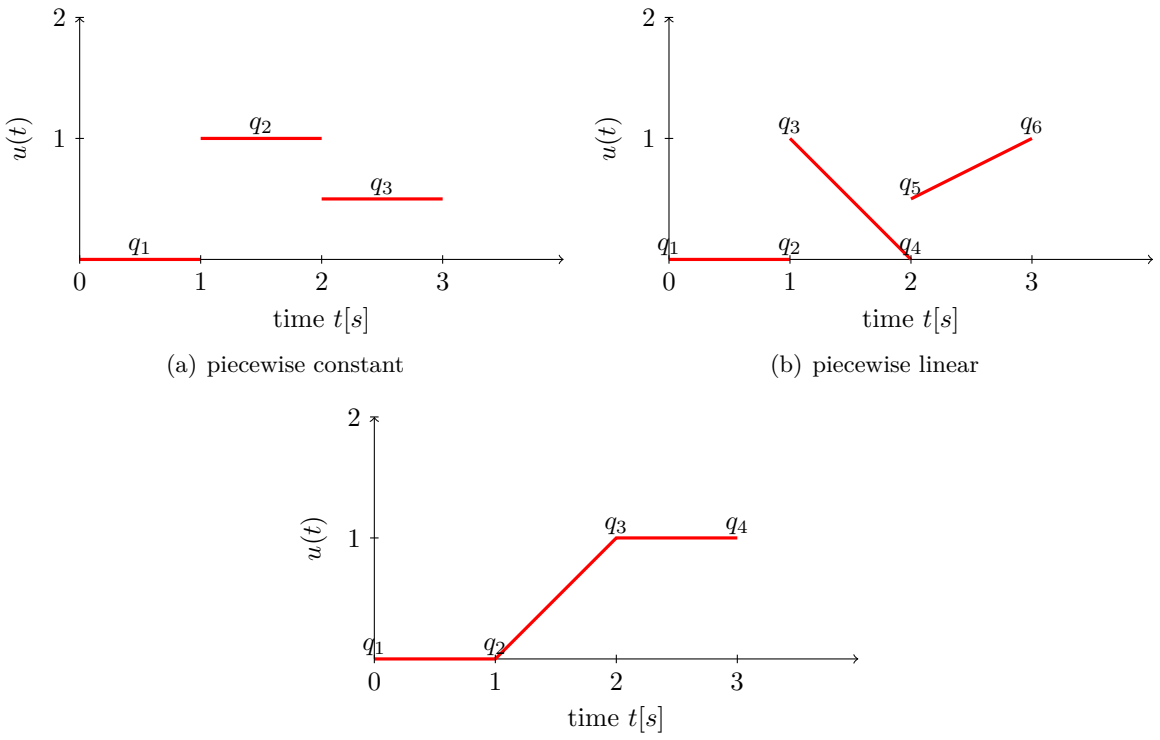
**Globally Constant** Certain controls, such as initial values, are are control variables and not functions. For notational simplicity, such control variables are regarded as globally constant control functions, i.e.,

$$u(t) = q \tag{4.10}$$

for all $t$.

## 4.4. Numerical Optimization with Dynamical Systems Constraints

Despite the fact that one can think of the state trajectory as a function $x(t; x_0, u, p)$ of the parameters $p$, control functions $u(t)$ and initial value $x_0$, its algorithmic evaluation is an iterative process akin to the solution of a large structured nonlinear system of equations. Similarly, optimization problems are commonly reformulated as nonlinear systems of equations by virtue of the KKT theorem. It is thus possible to combine both tasks. There are various degrees how strongly these distinct tasks are coupled and there is a trade-off between potential numerical efficiency and "ease of use". See Figure 4.2 for a graphical interpretation.




(a) piecewise constant

(b) piecewise linear

(c) piecewise linear continuous

Figure 4.1.: This figure shows different control function parametrizations. The position of the $q$ values correspond to the numerical value. E.g., the piecwise constant control function $u(t)$ is parametrized by $q = [q_1, q_2, q_3] = [0, 1, 0.5]$. The switching times are $t_{\text{ust}} = [0, 1, 2, 3]$.

In the *sequential approach* one uses an integration scheme to compute the state. When the state has been evaluated with sufficient accuracy, it is subsequently used to evaluate the objective function of the nonlinear program. The advantage of the sequential approach is that at each iterate of the optimization one obtains a state trajectory that allows an interpretation. I.e., each iterate represents a physically meaningful state. One can describe the sequential approach by a nonlinear program of the form

$$\begin{aligned} &\min_{p \in \mathbb{R}^{N_p}} f(x(t; x_0(u,p), u, p), p) \\ \text{where} \quad &0 = M(t, x, u, p) \\ &0 = x(t_0) - x_0(u,p) \quad \text{is satisfied by} \quad x(t; x_0(u,p), u, p) \quad \text{for } t \in [t_0, t_e] \,. \end{aligned}$$

By "where" it is meant that $x(t; x_0(u,p), u, p)$ is regarded as a function of $u$ and $p$ satisfying the model equations and should not be confused with s.t. (subject to).

To improve the numerical properties of the solution process one can couple the integration and optimization algorithm. Very popular is for instance *multiple shooting* where the time horizon is partitioned into disjoint time intervals. Additional matching conditions are included as equality constraints to the optimization problem, see for instance the work by Bock [1987] and Bulirsch [1971]. Such methods are called *simultaneous* in this thesis (c.f. Schäfer [2004]). The solution of the DAE on each subinterval is still considered to be a function of initial values,

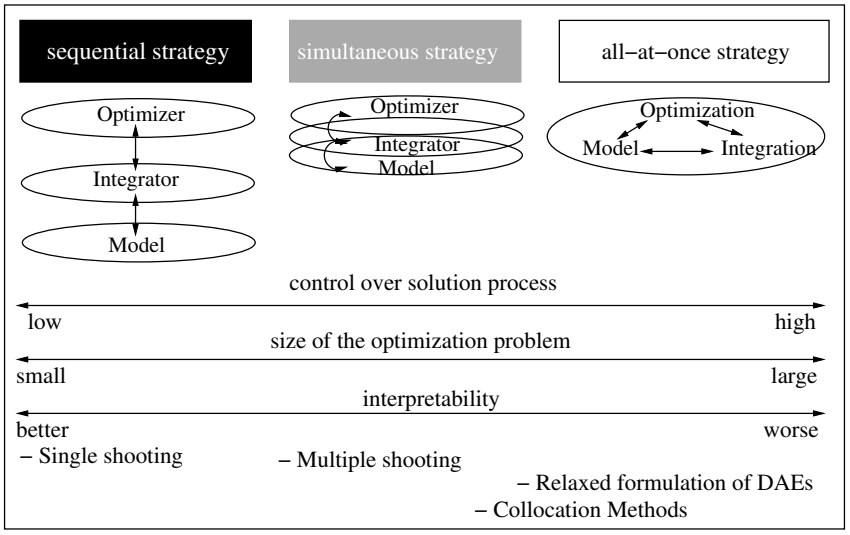



Figure 4.2.: A graphical comparison between the sequential, simultaneous and the all-at-once approach. The integrator and the model are decoupled in a sequential approach. In terms of software this means that the model, integrator and optimizer can be written by different persons. I.e., for the optimizer one could use an available SQP tool, for the integration some integration scheme which supports evaluation of derivatives and the model is written by a chemical engineer. In a simultaneous strategy one exploits the inherent structure of the dynamical system and gets a more fine-grained control over the solution process. A typical example is multiple shooting which results in an optimization problem with special KKT systems. When the evaluation of the model itself is expensive and requires an iterative process it can be advantageous to couple the simulation with the optimization in an all-at-once fashion.

parameters and control functions. Formally, the multiple shooting optimization problem is

$$\begin{aligned}
\min_{s_1\in\mathbb{R}^{N_x},\ldots,s_{N_{\mathrm{ms}}}\in\mathbb{R}^{N_x},p\in\mathbb{R}^{N_p}} & f(x(t;(s_1,\ldots,s_{N_{\mathrm{ms}}}),u,p),p)\\
\text{s.t.}\quad 0 =& x(t_i;s_i,u,p)-s_{i+1}\ ,i=1,\ldots,N_{\mathrm{ms}}-1\\
0 =& s_1-x_0(u,p)\\
\text{where}\quad 0 =& M(t,x,u,p) \text{ is satisfied by } x(t;s_i,u,p) \text{ for } t\in(t_{i-1},t_i]\ ,i=1,\ldots,N_{\mathrm{ms}}.
\end{aligned}$$

Hence, the solution trajectory may just be feasible in the solution point.

It is also possible to relax conditions which define the model and include them as equality constraints of the optimization problem. Consider for instance

$$\begin{aligned}
\min_{p\in\mathbb{R}^{N_p}} & f(x(t;x_0(u,p),u,p),p)\\
\text{s.t.}\quad 0 =& g(t_0,y_0,z_0,u(t_0),p)\\
\text{where}\quad A(t,y,z,p)\dot{y} =& f(t,y,z,u,p)\\
0 =& g(t,y,z,u,p)-\beta(t)g(t,y_0,z_0,u(t_0),p)\\
y(t_0) =& y_0(p,u)\\
z(t_0) =& z_0 \quad \text{is satisfied by} \quad x(t;x_0,u,p)\ .
\end{aligned} \tag{4.11}$$

As described by Bock et al. [1988] the consistency of the model is *relaxed* and used as equality constraint of the nonlinear program. I.e., when an infeasible set optimizer is used, the feasibility

$0 = g(t_0, y_0, z_0, u, p)$ of the model is not satisfied in each iteration of the optimization process. The *damping function* $\beta : \mathbb{R} \to \mathbb{R}$ is a non-negative, strictly decreasing function with $\beta(t_0) = 1$, e.g. $\beta(t) = e^{-\alpha(t-t_0)}$, $\alpha > 0$. The algebraic equations are thus always satisfied and therefore any $z_0$ is a consistent initial value of the DAE.

The numerical solution of large-scale PDEs often requires an iterative solution of nonlinear systems. Since also the optimization process is of iterative nature one can try to couple both iterations to reduce the net number of cycles. One possibility is to write the PDE as constraint of the optimization problem, i.e., the optimization problem reads

$$\begin{aligned} \min_{x,p \in \mathbb{R}^{N_p}} & f(x,p) \\ \text{s.t.} \quad 0 =& M(t,x,u,p) \\ 0 =& x(t_0) - x_0(u,p) \ . \end{aligned}$$

This is approach is called *all-at-once* in this thesis. In other words, the whole model is defined as a constraint of the optimization problem and thus, depending on the optimization strategy, the model may be be unphysical/infeasible during the optimization. Treating the whole problem all-at-once has the advantage that one can manually tune the solution process.

# 5. Parameter Estimation

Mathematical models in science and engineering are often of dynamical nature. Their derivation involves theoretical considerations such symmetries and conservation laws. Although the underlying physics are extremely well validated, complex interactions between otherwise simple systems can produce new, and sometimes surprising, behavior with a complexity impossible to tackle. Fortunately, the complex behavior can often be described by derived models that are of simple mathematical form. This process of simplification often involves statistical approximations. For instance, one describes a thermodynamic system by notions such as temperature and pressure, and not as an ensemble of atoms. The price one pays is that such models contain *parameters* $p \in \mathbb{R}^{N_p}$ that are known to exist but are of unknown exact value. It is therefore necessary to perform an experiment with the goal to obtain numerical values for $p$. However, it is generally not possible to measure the parameters in a direct fashion. Rather, one can only measure quantities that depend on the *state* $x$ of the model. Additionally, the experimental setup influences how a measurement $\eta$ can be observed. One is confronted with an inverse problem where it can happen that the experimental design is inadequate to infer the underlying parameters.

Furthermore, random errors appear in the measurement process. Hence, finding the true parameters is also influenced by information loss due to noise. An experimenter sets up an experiment and tries to infer from the measured data $\eta$ an estimate $p(\hat{\eta}(\omega))$ of the parameters, where $\omega$ is a sample as explained in Chapter 5.1. Since the estimate depends on the measurements and the measurements are of uncertain nature, one obtains only a guess for the true parameters. One says that the errors in the measurements propagate to an error in the parameters. It is therefore mandatory to state how good an estimate is. A useful quantity are confidence sets because they allow an easy interpretation. They are introduced in Chapter 5.1.

To add another layer of information loss, these models are evaluated on a digital computer. It is rather common that numerical problems due to finite precision arithmetic arise when the system is ill-defined. Figure 5.1 visualizes the above explanation.

## 5.1. Elements from Mathematical Statistics

A statistician is supposed to give precise answers to questions in science, politics and economy based on observable quantities. Since there is randomness in the observed quantities it is generally not possible to give definitive answers. However, it is possible to quantify the probability that certain answers are correct or not. The purpose of this section is to collect the most important definitions from statistics and decision theory that are necessary to describe the statistical model used in the following sections. The discussion is largely based on Shao [2003], Dudley [2003], Künsch [2005] and the references therein.

### 5.1.1. Probability Theory

It is assumed that there is some set $\Omega$ called the *sample space*. A sample often includes several observed quantities. E.g., when a measurement of a real-valued quantity is repeated $N$ times, then the sample is $\omega = (\omega_1, \ldots, \omega_N) \in \mathbb{R}^N$. An element $\omega$ of $\Omega$ is called *sample* or *observation*. Subsets of the sample space $\Omega$ are called *events*. Depending on the the task, one defines subsets of the sample space that form a $\sigma$-algebra $\mathcal{A}$. For instance, $\{\omega : 0 \leq \omega \leq 1\}$ or

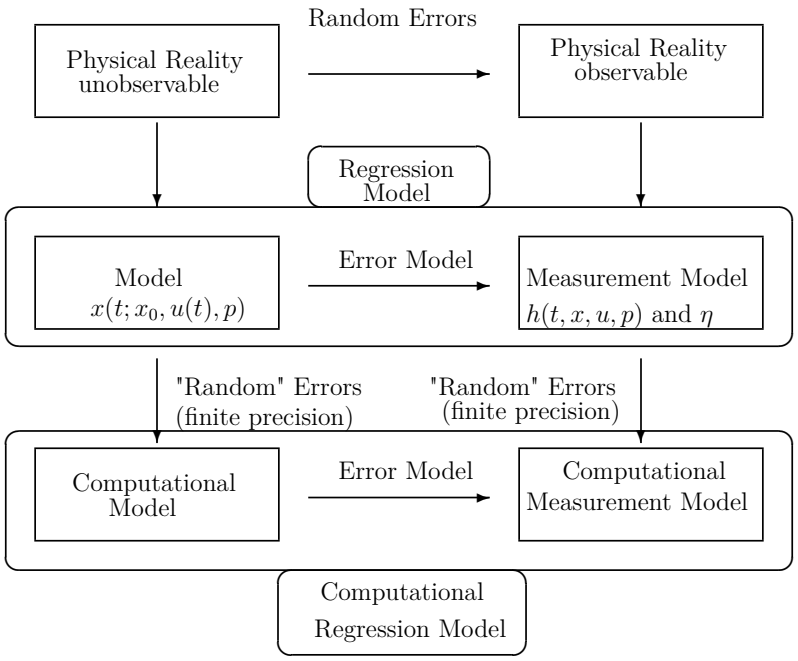



Figure 5.1.: The physical reality is modeled by a state $x(t; x_0, u(t), p)$ and the measurement process, which introduces additional errors, is described by a regression model. Additional errors occur in the numerical solution process.

$\{\omega : 0 \leq \sum_{n=1}^{N} \omega_n \leq 10\}$ could be events. One says that an event has *happened* if the outcome $\omega$ is an element of the subset of the associated event.

**Definition 5.1.1** (sigma algebra)**.** Let $\mathcal{A}$ be a collection of subsets of the sample space $\Omega$. $\mathcal{A}$ is called a $\sigma$-algebra (also called $\sigma$-field) if and only if it has the following properties:

(i) The empty set $\emptyset \in \mathcal{A}$.

(ii) If $A \in \mathcal{A}$ then so is the complement $A^c \in \mathcal{A}$.

(iii) If $A_i \in \mathcal{A}$, $i = 1, 2, \ldots$, then their union $\bigcup_i A_i \in \mathcal{A}$.

**Definition 5.1.2** (measure/probability measure)**.** A *measure* is a function

$$\begin{aligned} \mu : \mathcal{A} &\to [0, \infty] \\ A &\mapsto \mu(A) \ , \end{aligned}$$

satisfying the axioms

1. $\mu(\emptyset) = 0$ and $0 \leq \mu(\Omega) \leq \infty$
2. $\mu\left(\bigcup_{i=1}^{\infty} A_i\right) = \sum_{i=1}^{\infty} \mu(A_i)$ if $A_i \cap A_j = \emptyset$ for $i \neq j$ (countable additivity).

If $\mu(\Omega) = 1$ then $\mu$ is called *probability measure* and denoted $P$ instead of $\mu$.

**Definition 5.1.3** (measurable space, probability space)**.** The triple $(\Omega, \mathcal{A}, \mu)$ is called *measure space*. In the special case of a probability measure, i.e., $\mu = P$, then the triple $(\Omega, \mathcal{A}, P)$ is called *probability space*.

**Definition 5.1.4** (measurable function, Borel measurable/Borel function)**.** Let $f : X \to Y$ be a mapping between measurable spaces with sigma-algebras $\mathcal{A}$ resp. $\mathcal{B}$. Then $f$ is called *measurable* if and only if the inverse mapping satisfies

$$f^{-1}(B) \in \mathcal{A} \quad \text{for all } B \in \mathcal{B} \ .$$

If $Y = \mathbb{R}$ and $\mathcal{B}$ the Borel sigma-algebra, then $f$ is called *Borel measurable* or a *Borel function* on $(\Omega, \mathcal{A})$. One also says $f$ is a Borel function w.r.t. $\mathcal{A}$.

**Definition 5.1.5** (indicator function)**.** The *indicator function* is defined as

$$\mathbf{I}_A(\omega) := \begin{cases} 1 & \omega \in A \\ 0 & \text{else} \end{cases}$$

for any $A \subset \Omega$.

**Definition 5.1.6** (simple function)**.** A *simple function* $f$ can be written as a linear combination

$$f(\omega) = \sum_{n=1}^{N} a_n \mathbf{I}_{A_n}(\omega) \ ,$$

where $a_n \in \mathbb{R}$ and $A_1, \ldots, A_N$ measurable sets on $\Omega$.

**Definition 5.1.7** (random variable, state space, outcome)**.** A *random variable* on $\Omega$ is a measurable function

$$\begin{aligned} x : \Omega &\to S \\ \omega &\mapsto x(\omega) \ , \end{aligned}$$

where $S$ is called the *state space*. The realization $x(\omega), \omega \in \Omega$, is called an *outcome* of the random variable $x$. In the following, no distinction between random variable, random vector and random matrix is made.

*Remark.* In the literature, the random variable is typically denoted by capital letters $X$ and the outcomes by $x$. For the following discussion the outcome will be written $x(\omega)$ and the random variable itself as $x$.

**Definition 5.1.8** (law/distribution of a random variable)**.** Let $x$ be a random variable on $\Omega$, then

$$P_x := P \circ x^{-1}$$

is called the *law* or *distribution* of $x$.

**Definition 5.1.9** (integral of simple functions)**.** Let $f$ be a simple function, then its *integral* is defined as

$$\int f \mathrm{d}\mu := \sum_{n=1}^{N} a_n \mu(A_n) \ .$$

The integral is well-defined if $f$ is nonnegative. When $a_1 = -1$, $a_2 = 1$ and $\mu(A_1) = \mu(A_2) = \infty$ then the integral is $\infty - \infty = NaN$ ($NaN$=Not a Number).

**Definition 5.1.10** (integral of Borel functions)**.** Let $f$ be a nonnegative Borel function and $S_f$ be the collection of all nonnegative simple functions $\phi$ satisfying

$$\phi(\omega) \leq f(\omega)$$

for all $\omega \in \Omega$. Then the integral w.r.t. $\mu$ is defined as

$$\int f \mathrm{d}\mu := \sup \left( \left\{ \int \phi \mathrm{d}\mu : \phi \in S_f \right\} \right) \ .$$

I.e., there exists a sequence of simple functions $\phi_1, \phi_2, \ldots$ such that $0 \leq \phi_i \leq f$ for all $i$ and $\lim_{i\to\infty} \int \phi_i \mathrm{d}\mu = \int f \mathrm{d}\mu$.

**Definition 5.1.11** (integrable)**.** Let $f$ be a Borel function. One says that $\int f \mathrm{d}\mu$ exists if either $\int \max(f, 0)\mathrm{d}\mu < \infty$ or $\int \max(-f, 0)\mathrm{d}\mu < \infty$. When both are finite the function $f$ is called *integrable*.

**Definition 5.1.12** (expectation value)**.** The *expectation value* of a random variable $x : \Omega \to S$ is defined as

$$\mathbb{E}[x] := \int x \mathrm{d}P \ .$$

**Definition 5.1.13** (covariance matrix)**.** Let $x : \Omega \to \mathbb{R}^N$ be a random variable. The *covariance matrix* of $x$ is defined as

$$\mathrm{Cov}(x) := \mathbb{E}[(x - \mathbb{E}x)(x - \mathbb{E}x)^T] \ .$$

**Definition 5.1.14** (cumulative distribution function)**.** Let $x : \Omega \to \mathbb{R}$ be a Borel measurable random variable on the probability space $(\Omega, \mathcal{A}, P)$ . The *cumulative distribution function* (cdf) of $P$ is defined as

$$F(y) := P(\{\omega \in \Omega : x(\omega) \leq y\}) \ .$$

The cumulative distribution function $F$ has to satisfy the notion of absolute continuity. However, it will not be necessary for the following discussion and in the following only the stronger condition of $F$ to be continuously differentiable is imposed. A treatment of the more general case can be found in Shao [2003]. Later on, the probability space will be defined by the probability density function. That means, not the distribution function has to be derived from the probability measure, but the other way around.

**Definition 5.1.15** (probability density function)**.** Let the cumulative distribution function $F : \mathbb{R} \to (0, 1)$ be continuously differentiable. One calls

$$f(y) := \frac{\partial F}{\partial y}(y)$$

the *probability density function* (pdf) of the distribution $F$. It satisfies

$$F(y) := \int_{-\infty}^{y} f(z)\mathrm{d}z \ ,$$

where the usual Riemann integral is used.

**Definition 5.1.16** (quantile function)**.** Let $x$ by a real-valued random variable with cumulative distribution function $F : \mathbb{R} \to (0, 1)$, then the inverse function

$$\begin{aligned} q_-^F(u) &:= \inf(\{y \in \mathbb{R} : F(y) \geq u\}) \\ q_+^F(u) &:= \sup(\{y \in \mathbb{R} : F(y) \leq u\}) \end{aligned}$$

are called the *quantile functions*.

### 5.1.2. Statistical Decision Theory

Statistical decision theory could be phrased as the problem of *choice under uncertainty*. Consider the random variable

$$\begin{aligned} x : \Omega &\to \mathbb{X} \\ \omega &\mapsto x(\omega) \ . \end{aligned}$$

It is advantageous to call $(\mathbb{X}, F_{\mathbb{X}})$ a *sample space* as well. $F_{\mathbb{X}}$ is a $\sigma$-algebra. This stems from that fact that since $x$ is assumed to be measurable, one can use the pullback $P(\{x : x \in A\}) = P(\{\omega : x(\omega) \in A\})$ to define a probability measure on $\mathbb{X}$. In that regard, both sample spaces are equivalent, and one sometimes switches back and forth.

In the following, it is assumed that the outcomes $\omega$ are distributed according to one probability distribution from the family $\{P_\theta : \theta \in \Theta\}$. The *statistical parameter* $\theta$ can be finite but also be infinite dimensional. It contains all information that is necessary to answer a problem or to arrive at a decision. For the following discussion the case $\Theta = \{(\mu, \sigma^2)\} = \mathbb{R} \times \mathbb{R}_+$ for $\omega \sim \mathcal{N}(\mu, \sigma^2)$ will be of interest.

**Definition 5.1.17** (action space)**.** The tuple $(\mathbb{A}, \mathcal{F}_{\mathbb{A}})$, where $\mathbb{A}$ is the set of all possible actions and $\mathcal{F}_{\mathbb{A}}$ a $\sigma$-algebra on $\mathbb{A}$, is called *action space*.

**Example 5.1.1** (action spaces)**.** Examples for action spaces are

1. $\mathbb{A} = \mathbb{R}$ for a point estimate
2. $\mathbb{A} = \{0, 1\}$ in hypothesis testing
3. $\mathbb{A} = \{\text{intervals } \in \mathbb{R}\}$ for confidence intervals
4. etc.

They define the possible actions when an outcome $x(\omega)$ of a random variable $x$ is observed.

**Definition 5.1.18** (decision rule)**.** A *decision rule* is measurable function from the sample space $(\mathbb{X}, \mathcal{F}_{\mathbb{X}})$ to the *action space* $(\mathbb{A}, \mathcal{F}_{\mathbb{A}})$, i.e.,

$$\begin{aligned} d : \mathbb{X} &\to \mathbb{A} \\ x(\omega) &\mapsto a(\omega) = d(x(\omega)) \ . \end{aligned}$$

Such a decision rule says that if $x(\omega)$ is observed then the action resp. decision $d(x(\omega))$ should be taken. When the probability distribution is described by a statistical parameter $\theta$ it follows that the decision depends implicitly on $\theta$.

**Example 5.1.2.**

- Sample space and distribution: Let $P_\theta = \mathcal{N}(\theta, \sigma^2)^N$ on $\Omega = \mathbb{R}^N$, where $-\infty < \theta < \infty$ and $\sigma$ is fixed and known. I.e., the outcomes $\omega_n$ are i.i.d. normally distributed.
- random variable: Consider the random variable $\hat{\theta}(\omega) := \frac{1}{N} \sum_{n=1}^{N} \omega_n$. It is the standard estimator for the unknown parameter $\theta \in \mathbb{R}$.
- decision space: Given the sample $\theta(\omega)$ one would to decide on a confidence interval. One can show that $[\hat{\theta}(\omega) - 1.96\frac{\sigma}{\sqrt{N}}, \hat{\theta}(\omega) + 1.96\frac{\sigma}{\sqrt{N}}]$ is a 95% confidence interval for $\theta$ in the sense that for all $\theta$ it holds that $P_\theta(\{\omega : |\hat{\theta}(\omega) - \theta| > 1.96\frac{\sigma}{\sqrt{N}}\}) = 0.05$. Since $\hat{\theta}(\omega)$ is random it also follows that the confidence region is random. See Figure 5.4 for a graphical explanation.

To be able to compare different decisions, a *loss function* $L$ is introduced.

**Definition 5.1.19** (loss function)**.** A function

$$\begin{aligned}\Theta \times \mathbb{A} \to &[0, \infty) \\ \theta, a(\omega) \mapsto &L(\theta, a(\omega)) \ .\end{aligned}$$

is called *loss function* if a "bad" decision $b(\omega) \in \mathbb{A}$ and a "good" decision $g(\omega) \in \mathbb{A}$ satisfy

$$L(\theta, b(\omega)) \geq L(\theta, g(\omega)) \ .$$

Loss functions are, just like decision rules, random variables. To be able to obtain a single number how "good" a taken decision is, a so-called *risk function* is introduced.

**Definition 5.1.20** (risk function)**.** Let the decision rule $d$ be fixed. The function

$$\begin{aligned}r_d : \Theta \to &[0, \infty) \\ \theta \mapsto &r_d(\theta) := \mathbb{E}[L(\theta, d \circ x)] = \int L(\theta, d(x)) \mathrm{d}P_\theta(x) \ ,\end{aligned}$$

is called the *risk* resp. *risk function*. I.e., the risk is the expected loss.

It is then possible to compare decision rules $f, g$ by their risk. Since the parameters $\theta$ are not known, there is typically no best decision rule. To obtain a risk that is independent of $\theta$ one can use the *minimax risk*

$$r_d := \sup_{\theta \in \Theta} r_d(\theta)$$

or the *Bayes risk*

$$r_d(\pi) := \int_{\theta \in \Theta} r_d(\theta) \mathrm{d}\pi(\theta)$$

It requires a "prior" $\pi$ on the parameters $\theta$.

**Definition 5.1.21** (derived parameter)**.** The numerical value of a function

$$\begin{aligned}g : \Theta \to &\Gamma \\ \theta \mapsto &g(\theta)\end{aligned}$$

is called a (derived) *parameter* in the following.

**Definition 5.1.22** (estimator)**.** An *estimator* is a function from the sample space $\mathbb{X}$ to another space $\Gamma$ and is denoted

$$\begin{aligned}\hat{g} : \mathbb{X} \to &\Gamma \\ x(\omega) \mapsto &\hat{g}(x(\omega)) \ .\end{aligned}$$

That means an estimator is a random variable $\hat{g}$ and more precisely a decision rule. For each sample $\omega$ one obtains a *sample estimate* $\hat{g}(x(\omega))$. Estimators for single values are called *point estimates* and can be seen as the counterpart to *set estimators* as for instance confidence sets.

**Example 5.1.3.** Let $x \sim \mathcal{N}(\theta, 1)$, then $\hat{\theta}(\omega) := \frac{1}{N} \sum_{n=1}^{N} x(\omega_n)$ is the classical estimator of $\theta$.

**Example 5.1.4.** Let $x : \Omega \to x(\omega)$ be a random variable, $g(\theta) \in \mathbb{R}$ a derived parameter and $\hat{g}(\cdot; \theta)$ a data-dependent estimator for $g(\theta) \in \mathbb{R}$. I.e., as decision function one can use $d(x(\omega)) := \hat{g}(x(\omega); \theta)$ and as loss function

$$L(\theta, d(x(\omega)) = w(\theta) |g(\theta) - d(x(\omega))|^r \ .$$

It states that the decision on the estimated parameter should be as close to the true parameter as possible. $w(\theta)$ is a weight for the statistical parameter $\theta$ and for instance $r = 2$.

**Definition 5.1.23** (statistical test)**.** A *(statistical) test* is a decision rule that maps to the integers. Popular tests are for instance hypothesis tests $d : \omega \mapsto \{0, 1\}$.

**Definition 5.1.24** (confidence set)**.** Let $(\Omega, \mathcal{A}, P_\theta)_{\theta\in\Theta}$ be a family of probability spaces and $x : \Omega :\to \mathbb{X}$ a random variable. The data dependent set $\mathrm{CR}(x(\omega))$ is called *confidence set* for the derived parameter $g(\theta)$ to the *level of significance* $1 - \alpha$, $\alpha \in (0, 1)$ if and only if

$$\inf_{\theta\in\Theta} \left(P_\theta \left(\{\omega : g(\theta) \in \mathrm{CR}(x(\omega))\}\right)\right) \geq 1 - \alpha \; . \tag{5.1}$$

More verbosely: if 100 samples $x(\omega_i)$ are drawn, then $g(\theta)$ is contained in about $(1 - \alpha)100$ of the associated confidence sets $\mathrm{CR}(x(\omega_i))$.

### Confidence Sets and the Covariance Matrix

**Definition 5.1.25** (normal distribution)**.** A random variable $x : \Omega \to \mathbb{R}$ is said to be *normally distributed* with mean $\mu \in \mathbb{R}$ and variance $\sigma^2 \in \mathbb{R}_+$ if it has the probability density function

$$f(x) = \frac{1}{(2\pi)^{N/2}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \; . \tag{5.2}$$

One writes $x \sim \mathcal{N}(\mu, \sigma^2)$. In the multivariate case $x : \Omega \to \mathbb{R}^N$

$$f(x) = \frac{1}{(2\pi)^{N/2}\det(\Sigma^2)^{1/2}} e^{-\frac{1}{2}(x-\mu)^T\Sigma^{-2}(x-\mu)} \; . \tag{5.3}$$

One writes $x \sim \mathcal{N}(\mu, \Sigma^2)$, where $\mu \in \mathbb{R}^N$ and $\Sigma^2 \in \mathbb{R}^{N\times N}$ is symmetric positive definite.

**Definition 5.1.26** (chi-square distribution)**.** Let $x \sim \mathcal{N}(0, \mathbf{1}_N)$ be a normally distributed random variable. The random variable

$$y(\omega) := \sum_{n=1}^{N} x_n(\omega) \tag{5.4}$$

is *chi-square distributed* with $N$ degrees of freedom.

**Lemma 5.1.1.** *Let $(\Omega, \mathcal{A}, P_\theta)_{\theta\in\Theta}$ be a family of probability spaces and $x : \Omega \to \mathbb{R}^N$ normally distributed, i.e., $x \sim \mathcal{N}(\mu, \Sigma^2)$, then*

$$\mathrm{CR}(x(\omega); \alpha) := \{y \in \mathbb{R}^N : (x(\omega) - y)^T\Sigma^{-2}(x(\omega) - y) \leq \gamma_N^2(\alpha)\} \tag{5.5}$$

*is a confidence set of $\mu$ to the level of significance $1 - \alpha$ and $\gamma_N^2(\alpha)$ is defined by $F_{\chi_N^2}(\gamma_N^2(\alpha)) = 1 - \alpha$. $F_{\chi_N^2}$ is the cumulative distribution function of the $\chi^2$ distribution with $N$ degrees of freedom.*

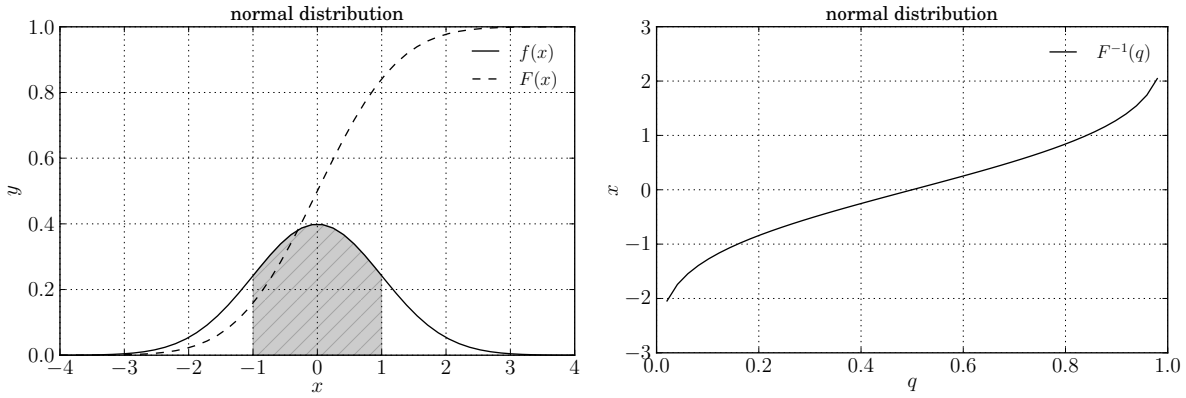



Figure 5.2.: In (a) one can see the probability density function $f(x)$ and cumulative distribution function $F(x)$ for $(\mu, \sigma^2) = (0, 1)$. The shaded area is the probability $0.68268949 = P_{(0,1)}(\{\omega : -\sigma \leq x(\omega) \leq \sigma\})$. I.e., the $x(\omega)$ is between $-1$ and 1 with probability $\approx 0.682$. In (b) the inverse cumulative distribution function is shown.

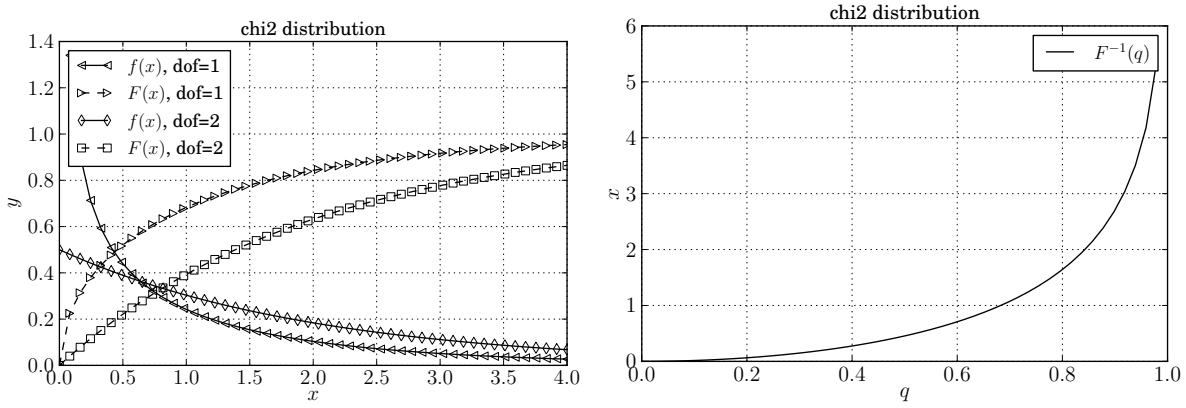



Figure 5.3.: In (a) one can see the probability density function $f(x)$ and the cumulative distribution function $F(x)$ for the $\chi^2$ distribution with one and two degrees of freedom (dof). In (b) the inverse cumulative distribution function for dof=1 is shown.

*Proof.* This follows from (5.1):

$$\begin{aligned}
P_\theta(\{\omega : \mu \in \mathrm{CR}(x(\omega))\} =& P_\theta(\{\omega : \mu \in \{y \in \mathbb{R}^N : (x(\omega) - y)^T \Sigma^{-2} (x(\omega) - y) \leq \gamma_N^2(\alpha)\}\}) \\
=& P_\theta(\{\omega : (x(\omega) - \mu)^T \Sigma^{-2} (x(\omega) - \mu) \leq \gamma_N^2(\alpha)\}) \\
=& P_\theta(\{\omega : Z(\omega)^T Z(\omega) \leq \gamma_N^2(\alpha)\}) \\
=& P_\theta(\{\omega : \sum_{n=1}^{N} Z_n^2(\omega) \leq \gamma_N^2(\alpha)\}) \\
=& F_{\chi_N^2}(\gamma_N^2(\alpha)) \\
=& 1 - \alpha \, .
\end{aligned}$$

The random variable $Z(\omega) := \Sigma^{-1}(x(\omega) - \mu)$ is $\mathcal{N}(0, \mathbf{1}_N)$ distributed and is therefore independent of $\theta$. That means that $\inf_\theta P_\theta(\{\omega : \mu \in \mathrm{CR}(x(\omega))\} = P_\theta(\{\omega : \mu \in \mathrm{CR}(x(\omega))\}$. The last step follows from the definition of $\gamma_N^2(\alpha)$. □

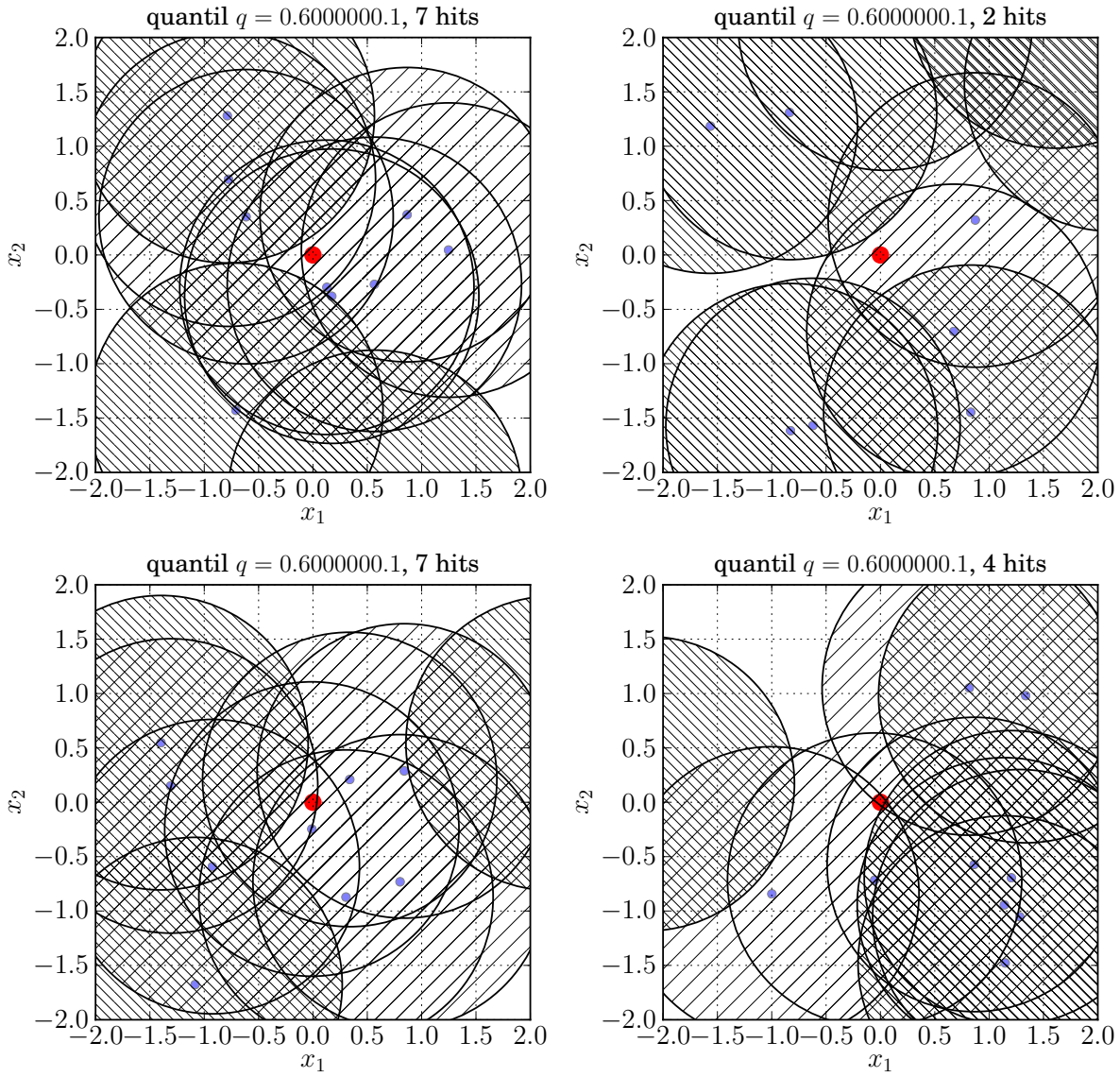



Figure 5.4.: All plots show confidence regions of a two-dimensional random variable $x \sim \mathcal{N}((0,0)^T, \mathbf{I}_2)$ to the level of significance $1 - \alpha = q = 0.6$. The ellipses with hatch / contain the true $\mu = (0,0)^T$, whereas the ellipses with hatch \ do not.

### Linear Error Propagation

Let $x \sim \mathcal{N}(\mu, \Sigma^2)$ be a random variable and $\mu \in \mathbb{R}^N$. Let $f : \mathbb{R}^N \to \mathbb{R}^M$ be an affine function, i.e., $f(x(\omega)) = Ax(\omega) + b$. It then follows that $y = f \circ x$ is $\sim \mathcal{N}(A\mu + b, A\Sigma^2 A^T)$ distributed. If $f \in C^1$ is nonlinear one can use a first-order Taylor expansion to obtain locally an affince approximation, i.e.,

$$f(x) = f(\mu) + \frac{\partial f}{\partial x}(x)\Big|_{x=\mu} (x - \mu) + o(|x - \mu|)$$

which can be written as $f(x) = Ax + b + o(|x - \mu|)$. It follows that the random variable is approximately $y \sim \mathcal{N}(f(\mu), J(\mu)\Sigma^2 J(\mu)^T)$ distributed, where $J(\mu) = \frac{\partial f}{\partial x}(x)\Big|_{x=\mu}$.

### 5.1.3. Maximum Likelihood and Least Squares Estimation

**Definition 5.1.27** (likelihood function, maximum likelihood estimate)**.** Let $x : \Omega \to \mathbb{X}$ be a random variable with probability density function $f_\theta$ w.r.t. a $\sigma-$finite measure, where $\theta \in \Theta \subseteq \mathbb{R}^N$.

1. For each $y = x(\omega)$, $f_\theta(y)$ as function of $\theta$ is called *likelihood function* and is denoted
$$l(\theta) := f_\theta(y) \ .$$
2. Let $\bar{\Theta}$ be the closure of $\Theta$. An estimate $\theta \in \bar{\Theta}$ satisfying
$$l(\hat{\theta}) = \max_{\theta \in \bar{\Theta}} l(\theta) \tag{5.6}$$
is called a *maximum likelihood estimate* of $\theta$.
3. Let $g$ be a Borel function from $\Theta$ to $\mathbb{R}^M$, $M \leq N$. If $\hat{\theta}$ is a maximum likelihood estimate of $\theta$, then one says $g(\hat{\theta})$ is a maximum likelihood estimate of $g(\theta)$.

Put into words, the maximum likelihood estimator tries to find $\theta$ such that the observed sample $x(\omega)$ would be most likely. The closure of $\Theta$ is used to guarantee the existence of a maximizer in the set.

If the probability density function $f_\theta$ is known one can often reformulate the maximum likelihood estimator. For instance, let $x \sim \mathcal{N}(\mu, \Sigma^2)$ and $\mu \in \mathbb{R}^N$ and consider the covariance matrix $\Sigma^2$ to be known and fixed. That means $\theta = \mu$ and the probability density function is

$$f_\mu(x(\omega)) = \frac{1}{(2\pi)^{N/2} \det(\Sigma^2)^{1/2}} e^{-\frac{1}{2}(x(\omega)-\mu)\Sigma^{-2}(x(\omega)-\mu)} \ . \tag{5.7}$$

Inserting this into the definition of the *maximum likelihood estimator* one obtains

$$\begin{aligned} \hat{\mu} =& \text{argmax}_{\mu \in \mathbb{R}^N} f_\mu(x(\omega)) = \text{argmax}_{\mu \in \mathbb{R}^N} \log f_\mu(x(\omega)) \\ =& \text{argmax}_{\mu \in \mathbb{R}^N} -\frac{1}{2}(x(\omega)-\mu)^T \Sigma^{-2}(x(\omega)-\mu) + \text{const.} \\ =& \text{argmin}_{\mu \in \mathbb{R}^N} \frac{1}{2}(x(\omega)-\mu)^T \Sigma^{-2}(x(\omega)-\mu) \\ =& \text{argmin}_{\mu \in \mathbb{R}^N} \frac{1}{2}\|\Sigma^{-1}(x(\omega)-\mu)\|_2^2 \ , \end{aligned} \tag{5.8}$$

This estimator is called *weighted least squares estimator*. Note that if the variance is unknown then the const. expression may not be neglected and gives rise to an maximum likelihood estimator for the variance.

## 5.2. Estimating Parameters in Dynamical Systems

Parameter estimation problems as they arise in chemical engineering depend on dynamical systems. Thus, no explicit solution is known in general. On the other hand, the dynamical nature of the models result in parameter estimation problems of specific structure. The purpose of this section is to specialize the discussion of the previous Chapter 5.1 to this special class of parameter estimation problems.

### 5.2.1. Regression Model

It is assumed that finitely many measurements are taken at the *measurement times*

$$\begin{aligned} t_{\text{mts}} :=& (t_1, \dots, t_{N_{\text{mts}}})^T \in \mathbb{R}^{N_{\text{mts}}} \\ & t_1 < t_2 < \cdots < t_{N_{\text{mts}}} \ , \end{aligned} \tag{5.9}$$

where the first and last measurement time satisfy $(t_1, t_{N_{\text{mts}}}) \subseteq [t_0, t_e]$, and mts is an abbreviation for *measurement times*. For notational simplicity, the notation

$$x(t_{\text{mts}}) := x(t_{\text{mts}}; x_0, u, p) := (x(t_1), x(t_2), \dots, x(t_{N_{\text{mts}}}))$$

is used, where

$$x(t) \equiv x(t; x_0, u, p) \in \mathbb{R}^{N_x} \ .$$

It must be emphasized that $t_i$ interpreted as a measurement time is unrelated to the initial time $t_0$. I.e., it is possible that $t_{n_{\text{mts}}} = t_0$ for $n_{\text{mts}} = 1$. The quantity that can be measured is denoted

$$h(x(t; x_0, u, p), p) \in \mathbb{R}^{N_h} \tag{5.10}$$

and is referred to as *measurement function* or *model response*. It is assumed that the measurement function $h$ depends only on the state $x(t)$ and the parameters $p$. I.e., control functions $u$ enter only through the state $x(t; x_0, u, p)$.

At each measurement time $t_{n_{\text{mts}}}$, $n_{\text{mts}} = 1, \dots, N_{\text{mts}}$, one sample

$$\eta_{n_{\text{mts}}} := W_{n_{\text{mts}}} h(x(t_{n_{\text{mts}}}; x_0, u, p), p) + \epsilon_{n_{\text{mts}}} \in \mathbb{R}^{N_{\eta_{n_{\text{mts}}}}} \tag{5.11}$$

can be observed. The equation describes how $h$ is related to the *measurement* $\eta_{n_{\text{mts}}}$, assuming additive errors $\epsilon \in \mathbb{R}^{N_{\eta_{n_{\text{mts}}}}}$. The matrix $W_{n_{\text{mts}}} \in \mathbb{R}^{N_{\eta_{n_{\text{mts}}}} \times N_h}$ defines how many measurements, if any at all, of the components in $h$ are measured at $t_{n_{\text{mts}}}$. It can be described by the weights vector $w \in \mathbb{N}_0^{N_h}$ through

$$(W_{n_{\text{mts}}})_{ij} = \delta_{\sum_{l=1}^{j-1} w_l < i \leq \sum_{l=1}^{j} w_l} \quad i = 1, \dots, \sum_{i=1}^{N_h} w_i \ , j = 1, \dots, N_h \ , \tag{5.12}$$

(see Example 5.2.1). The model assumption is that $\epsilon$ is a multivariate normally distributed random vector, i.e., $\epsilon \sim \mathcal{N}(0, \Sigma^2)$. Generally, not only one sample $\eta = \eta_{n_{\text{mts}}}$ is used but rather all measurements are combined into the *regression model*

$$\eta = \underbrace{\begin{pmatrix} W_1 & & \\ & \ddots & \\ & & W_{N_{\text{mts}}} \end{pmatrix}}_{=:W \in \mathbb{R}^{N_\eta \times N_{\text{mts}} N_h}} h(x(t_{\text{mts}}; x_0, u, p), p) + \epsilon \ , \tag{5.13}$$

where $N_\eta := \sum_{n_{\text{mts}}=1}^{N_{\text{mts}}} N_{\eta_{n_{\text{mts}}}}$ is the combined number of measurements at all measurement times. The model responses $h(x(t_{\text{mts}}; x_0, u, p), p)$ are vertically stacked vectors and is therefore of dimension $N_{\text{mts}} N_h$. One cannot observe the statistical error $\epsilon$ but only the residual

$$\tilde{F}_1(x(t_{\text{mts}}; x_0, u, p), p; \eta) := W \begin{pmatrix} h(x(t_1; x_0, u, p), p) \\ \vdots \\ h(x(t_{N_{\text{mts}}}; x_0, u, p), p) \end{pmatrix} - \begin{pmatrix} \eta_1 \\ \vdots \\ \eta_{N_{\text{mts}}} \end{pmatrix} \in \mathbb{R}^{N_\eta} \ , \tag{5.14}$$

where $W = W(w)$ is defined as above by the combined weights vector $w \in \mathbb{N}_0^{N_{\text{mts}} \times N_h}$. Later on,

one uses $F_1 = \Sigma^{-1}\tilde{F}_1$ which explains the use of the tilde. The action of measuring $\eta \in \mathbb{R}^{N_\eta}$ is called an *experiment*. For instance in Figure 5.5 one can see the measurements of one experiment with the error bars indicating the standard deviation and the simulated measurement model.

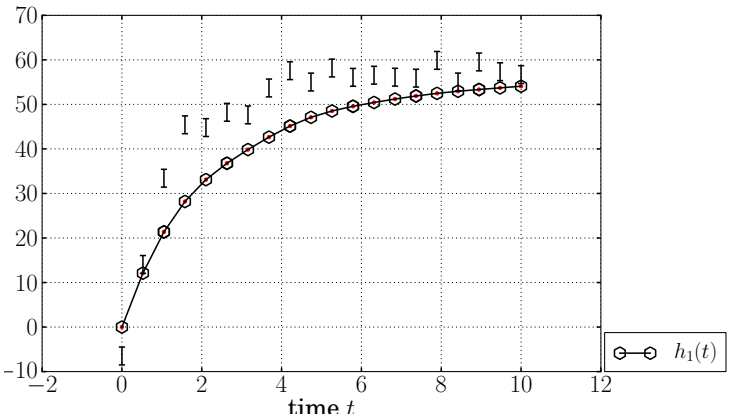



Figure 5.5.: This figure shows a plot of the measurement function $h(x(t_{\text{mts}}; x_0, u, p), p)$ of a dynamical system and observed values of the measurements. In total, measurements can be taken at the 20 measurement times $t_{\text{mts}}$. The red dots (in the marker symbols) indicate the weight of a measurement and is one for all measurement times. One can see that the measurement function does not describe the measurements very well.

**Example 5.2.1.** Let $N_h = 3$ and $w = (3, 0, 1)$ at the first measurement time $t_1$, then

$$W_1 = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \in \mathbb{R}^{4\times 3} \,. \tag{5.15}$$

As described in Chapter 5.1.3, a maximum likelihood estimation of a regression model with additive Gaussian errors can be formulated as a least squares problem. In practice, it may happen that the parameters $p$ are not independent. This leads to a constrained least squares problem or there are additional constraints. One encounters such situations when certain quantities, e.g., concentrations of substances, are necessarily positive. For a discussion see [Schlöder, 1987, Chapter I.3.1]. Equality constraints can also occur when *key performance indicators* are modeled as additional implicitly defined parameters (c.f. Chapter 5.2.4 and Körkel et al. [2008]) or, as remarked by Kostina et al. [2009], in PDE constrained optimization.

**Definition 5.2.1** (Initial Value Nonlinear Constrained Least Squares of Dynamical Systems)**.** Let $x(t; x_0, u, p) \in \mathbb{R}^{N_x}$ be the unique state trajectory defined by the initial values $x_0 \in \mathbb{R}^{N_x}$, control functions $u \equiv u(t) \in \mathbb{R}^{N_u}$ and parameters $p \in \mathbb{R}^{N_p}$. The nonlinear program

$$\begin{aligned} p_* = \operatorname{argmin}_{p\in\mathbb{R}^{N_p}} & \frac{1}{2}\|F_1(x(t_{\text{mts}}; x_0, u, p), p; \eta)\|_2^2 \\ \text{s.t.} \quad & 0 = F_2(t_{\text{cstr}}, x(t_{\text{cstr}}; x_0, u, p), p) \\ \text{where} \quad & 0 = M(t, x, u, p) \\ & 0 = x(t_0) - x_0 \text{ is satisfied by } x(t; x_0, u, p) \ . \end{aligned} \tag{5.16}$$

is called *initial value constrained least squares* problem in the following to stress that the initial values are known explicitly.

More generally, it may be the case that the initial values are not explicitly known but only *implicitly defined* by boundary value conditions $0 = r(t_{\text{cstr}}, x(t_{\text{cstr}}, p))$. These problems occur for instance when the system is time-periodic.

**Definition 5.2.2** (Boundary Value Nonlinear Constrained Least Squares of Dynamical Systems)**.** Let $x(t; x_0, u, p) \in \mathbb{R}^{N_x}$ be the unique state trajectory defined by the initial values $x_0 \in \mathbb{R}^{N_x}$, control functions $u \equiv u(t) \in \mathbb{R}^{N_u}$ and parameters $p \in \mathbb{R}^{N_p}$. The NLP

$$\begin{aligned} p_*, x_0 = \text{argmin}_{p\in\mathbb{R}^{N_p}, x_0\in\mathbb{R}^{N_x}} & \frac{1}{2}\|F_1(x(t_{\text{mts}}; x_0, u, p), p; \eta)\|_2^2 \\ \text{s.t.} \quad & 0 = \tilde{F}_2(t_{\text{cstr}}, x(t_{\text{cstr}}; x_0, u, p), p) \\ & 0 = r(t_{\text{cstr}}, x(t_{\text{cstr}}; x_0, u, p) \\ \text{where} \quad & 0 = M(t, x, u, p) \\ & 0 = x(t_0) - x_0 \text{ is satisfied by } x(t; x_0, u, p) \end{aligned} \tag{5.17}$$

is called *boundary value constrained least squares* problem in the following to stress that the initial values are only known implicitly. The residual $F_1$ is defined by

$$\begin{aligned} F_1(x(t_{\text{mts}}; x_0, u, p), p; \eta) :=& \Sigma^{-1}\tilde{F}_1(x(t_{\text{mts}}; x_0, u, p), p; \eta) \\ =& \Sigma^{-1}\left(W\begin{pmatrix} h(x(t_1; u, x_0, p), p) \\ \vdots \\ h(x(t_{N_{\text{mts}}}; x_0, u, p), p) \end{pmatrix} - \begin{pmatrix} \eta_1 \\ \vdots \\ \eta_{N_{\text{mts}}} \end{pmatrix}\right) \in \mathbb{R}^{N_\eta} , \end{aligned}$$

where $\eta \in \mathbb{R}^{N_\eta}$ are observed measurements distributed according to

$$\eta(\omega) \sim \mathcal{N}(h(x(t_{\text{mts}}; u, x_0, p_{\text{true}}), p_{\text{true}})), \Sigma^2)$$

and $W \in \mathbb{R}^{N_\eta \times N_{\text{mts}} N_h}$ the weight matrix. $\tilde{F}_2$ describe some relation between the parameters which are evaluated at some *constraint times* $t_{\text{cstr}} = (t_1, \ldots, t_{N_{\text{cstr}}})$. For convenience one can combine $\tilde{F}_2$ and $r$ as one system of equality constraints $F_2$.

In the following, only the boundary value nonlinear constrained least problem is considered. That means, it is necessary to optimize the parameters $p$ as well as initial values $x_0$. For ease of notation the following notation is used

$$v := (x_0, p) \in \mathbb{R}^{N_v} . \tag{5.18}$$

The boundary value constraints $r$ and constraints $\tilde{F}_2$ are combined in $F_2$. The efficient solution of this optimization problem requires the exploitation of its inherent structure. See the discussion by Schlöder [1987] or Bock [1987] for an in-depth treatise.

### 5.2.2. Linearization of the Solution Operator of the Parameter Estimation

The estimate $v(\eta(\omega))$ of the parameters $v$ depend on the observed measurement $\eta(\omega)$. That means the solution operator of the nonlinear program is an estimator $\hat{v} := v \circ \hat{\eta}$ for the true parameters $v$. In the nomenclature introduced in Chapter 5.1 the estimator $\hat{v}$ is a derived parameter $\hat{v} = v \circ \hat{\eta}$, where $\hat{\eta} \equiv \eta$ is an estimator for the mean $\mathbb{E}[\eta]$. In formulas,

$$\begin{aligned} \hat{v} : \Omega \to & \mathbb{R}^{N_v} \\ \omega \mapsto & \hat{v}(\omega) = v(\hat{\eta}(\omega)) . \end{aligned} \tag{5.19}$$

By looking at this equation it is clear that a repetition of the experiment results in another numerical value of the parameter. One would like to have a confidence region of the parameters

$v$. As discussed in Chapter 5.1.2, one can linearize the solution operator to obtain a first-order approximation of the confidence region. The Jacobian of the solution operator $v \circ \hat{\eta}$ is

$$J^+ := \frac{\partial \hat{v}}{\partial \eta}(\eta)\Big|_{\eta=\eta(\omega)} \in \mathbb{R}^{N_v \times N_\eta} . \tag{5.20}$$

One can show that $J^+$ is a generalized inverse of the matrix $J = (J_1^T, J_2^T)^T$, where $J_1(v) := \frac{\partial F_1}{\partial v}(v;\eta)$ and $J_2(v) := \frac{\partial F_2}{\partial v}(v)$. It satisfies $J^+JJ^+ = J$ but doesn't satisfy all Moore-Penrose axioms as it is the case for unconstrained least squares problems.

The solution operator is implicitly defined by the nonlinear program (5.17). Using the Karush-Kuhn-Tucker (KKT) conditions one can state the nonlinear program as solution of a nonlinear system of equations, see for instance the book by [Bonnans et al., 1997, Eqn. 13.1.] or the discussion by Schäfer [2004]. The application of the KKT theory to the linear error propagation has been treated by Bock et al. [2007b].

**Proposition 5.2.1** (Karush-Kuhn-Tucker Conditions)**.** *Consider the nonlinear program*

$$\begin{aligned} x_* = \operatorname{argmin}_{x\in\mathbb{R}^N} f(x) \\ s.t. \quad 0 = g(x) , \end{aligned} \tag{5.21}$$

*where $f : \mathbb{R}^N \to \mathbb{R}$ and $g : \mathbb{R}^N \to \mathbb{R}^M$ are twice continuously differentiable. Let the Lagrangian function be defined as*

$$\mathcal{L}(x, \lambda) := f(x) + \lambda^T g(x) \tag{5.22}$$

*with $\lambda \in \mathbb{R}^M$. If the Jacobian $\frac{\partial g}{\partial x}(x_*)$ has rank $M$ and $x_*$ is a solution of (5.21) then there exists a Lagrange multiplier $\lambda_* \in \mathbb{R}^M$ such that the tuple $(x_*, \lambda_*)$ satisfies the following equations*

$$\begin{aligned} 0 &= \nabla_x \mathcal{L}(x_*, \lambda_*) \quad \textit{optimality equation} \\ 0 &= g(x_*) \quad \textit{feasibility condition} . \end{aligned} \tag{5.23}$$

*They are called* first order necessary conditions *for optimality and $(x_*, \lambda_*)$ is called a* KKT point*. After application of the chain rule one finds that (5.23) reads*

$$0 = \begin{pmatrix} \nabla_x f(x_*) + (\lambda_*^T \frac{\partial g}{\partial x}(x_*))^T \\ g(x_*) . \end{pmatrix} \tag{5.24}$$

**Proposition 5.2.2** (Second Order Sufficient Conditions, SOSC)**.** *Let the Jacobian $J = \frac{\partial g}{\partial x}(x_*) \in \mathbb{R}^{M\times N}$ have rank $M$ and $x_*, \lambda_*$ be a KKT point. If $\nabla_x^2 \mathcal{L}(x_*, \lambda_*)$ is positive definite on the nullspace $Z \in \mathbb{R}^{N\times N-M}$ of $J$, i.e.,*

$$Z^T \nabla_x^2 \mathcal{L}(x_*, \lambda_*) Z = Z^T \left( \nabla_x^2 f(x_*) + \nabla_x (\lambda_*^T \frac{\partial g}{\partial x}(x_*)) \right) Z \geq 0 . \tag{5.25}$$

*then it follows that $x_*$ is a solution of the nonlinear program (5.21). The positive definiteness, together with* $\operatorname{rank}(J) = M$ *and the KKT condition is therefore a* second order sufficient condition *(SOSC) for optimality.*

Now, back to the previously introduced notation. The SOSC define a local minimum and therefore can be used to characterize how the parameter estimate changes when the observed sample is varied. It is useful to introduce the following notation.

**Definition 5.2.3** (LICQ)**.** One says that LICQ holds if

$$\operatorname{rank}(J_2(v)) = N_{F2} . \tag{5.26}$$

**Definition 5.2.4** (PD)**.** One says *positive definiteness* (PD) holds if

$$\operatorname{rank}(J(v)) = N_v \ , \tag{5.27}$$

where $J = (J_1^T, J_2^T)^T$.

**Lemma 5.2.3.** *Let LICQ and PD be satisfied. Then*

1. $J_1^T J_1$ *is positive definite on the nullspace of* $J_2$
2. *and* $M = \begin{pmatrix} J_1^T J_1 & J_2^T \\ J_2 & 0 \end{pmatrix}$ *is nonsingular.*

*Proof.* 1. Let $p \in \{p : p \neq 0, J_2 p = 0\}$. Since $(J_1^T, J_2^T)^T p \neq 0$ due to PD it follows that $J_1 p \neq 0$ and therefore $p^T J_1^T J_1 p \neq 0$ which is the definition of positive definiteness.

2. By contradiction: Let $p = (p_1^T, p_2^T)^T$ s.t. $Mp = 0$, i.e., also $J_2 p_1 = 0$, then it also follows that $0 = p^T M p = p_1^T J_1^T J_1 p_1 + p_2 J_2^T p_1 + p_2 J_2 p_1 = p_1^T J_1^T J_1 p_1 \neq 0$ which holds because of 1. This is a contradiction to the assumption.

□

**Proposition 5.2.4.** *Let LICQ and PD be satisfied, then the linearized solution operator of (5.16) (resp. (5.17)) is uniquely defined by*

$$\begin{aligned} J^+ =& P M^{-1} \begin{pmatrix} J_1^T \Sigma^{-1} \\ 0 \end{pmatrix} \in \mathbb{R}^{N_v \times N_\eta} \ , \\ \textit{where} \quad P :=& (\mathbf{I}_{N_v}, 0_{N_v, N_{F2}}) \\ M :=& \begin{pmatrix} J_1^T J_1 & J_2^T \\ J_2 & 0 \end{pmatrix} \ , \\ \textit{and} \quad J_1 =& \Sigma^{-1} W \frac{\partial h}{\partial v}(x(t_{\text{mts}}; x_0, u, p), p) \in \mathbb{R}^{N_\eta \times N_v} \ , \\ J_2 =& \frac{\partial F_2}{\partial v}(v) \in \mathbb{R}^{N_{F2} \times N_v} \ . \end{aligned}$$

*Above,* $0_{N_v, N_{F2}} \in \mathbb{R}^{N_v \times N_{F2}}$ *is a matrix where all elements are zero and* $\mathbf{I}_{N_v} \in \mathbb{R}^{N_v \times N_v}$ *is the identity matrix. Remember that* $v = (x_0, p)$*.*

*Proof.* This proof is given in similar form by Bock et al. [2007b] and makes use of the KKT first,- and second order optimality conditions, Lemma 5.2.3 and the definition of the regression model (5.14). In the used notation, the first order necessary conditions for optimality state

$$(a) \nabla_v \mathcal{L}(v_*, \lambda_*) = 0 \quad \text{optimality equation} \tag{5.28a}$$

$$(b) F_2(v_*) = 0 \quad \text{feasibility condition} \ , \tag{5.28b}$$

where the Lagrangian is defined as $\mathcal{L}(v, \lambda) = \frac{1}{2}\|F_1(v; \eta)\|_2^2 + \lambda^T F_2$. Hence, the KKT system is

$$0 = G(v, \lambda; \eta) = \begin{pmatrix} \nabla_v \mathcal{L}(v, \lambda; \eta) \\ F_2(v) \end{pmatrix} = \begin{pmatrix} J_1(v)^T F_1(v; \eta) + J_2(v)^T \lambda \\ F_2(v) \end{pmatrix} \ ,$$

where $J_1(v) := \frac{\partial F_1}{\partial v}(v; \eta)$ and $J_2(v) := \frac{\partial F_2}{\partial v}(v)$ haven been used. The $\eta$ are the inputs and $v, \lambda$ are the outputs. To find how a perturbation of $\eta$ changes $v$ and $\lambda$, a first order Taylor polynomial is propagated through this implicit system. I.e., to obtain the linearization, the univariate Taylor polynomial

$$\eta(T) = \underbrace{W h(x(t_{\text{mts}}, v_{[0]}), v_{[0]})}_{\eta_{[0]}} + \eta_{[1]} T$$

is propagated. For the *zeroth coefficient* one obtains from the regression model that $0 = F_1(v_{[0]}; \eta_{[0]})$. Using the information in $0 = G(v_{[0]}, \lambda_{[0]}; \eta_{[0]})$ one concludes that $0 = J_2^T \lambda_{[0]}$ has to hold. Because of LICQ it is therefore $\lambda_{[0]} = 0$. For the *first coefficient* one finds

$$0 = \frac{\partial}{\partial T} G(v(T), \lambda(T); \eta(T))\Big|_{T=0} = \partial_{(v,\lambda)} G(v_{[0]}, \lambda_{[0]}; \eta_{[0]}) \begin{pmatrix} v_{[1]} \\ \lambda_{[1]} \end{pmatrix} + \partial_\eta G(v_{[0]}, \lambda_{[0]}; \eta_{[0]})\, \eta_{[1]} ,$$

where $\eta_{[0]} := \eta(0)$ and $\eta_{[1]} := \frac{\partial}{\partial T}\eta(T)\Big|_{T=0}$. Hence,

$$\begin{pmatrix} v_{[1]} \\ \lambda_{[1]} \end{pmatrix} = - \left(\partial_{(v,\lambda)} G(v_{[0]}, \lambda_{[0]}; \eta_{[0]})^{-1} (\partial_\eta G)\, \eta_{[1]} .$$

The closed form expressions for $\partial_v G(v_{[0]}, \lambda_{[0]}; \eta_{[0]})$ and $\partial_\eta G(v_{[0]}, \lambda_{[0]}; \eta_{[0]})$ are given by

$$\partial_{(v,\lambda)} G(v_{[0]}, \lambda_{[0]}, \eta_{[0]}) = \begin{pmatrix} J_1^T J_1 & J_2^T \\ J_2 & 0 \end{pmatrix} ,$$
$$\partial_\eta G(v_{[0]}, \lambda_{[0]}, \eta_{[0]}) = \begin{pmatrix} -J_1^T \Sigma^{-1} \\ 0 \end{pmatrix} .$$

Due to Lemma 5.2.3 $\partial_{(v,\lambda)} G(v_{[0]}, \lambda_{[0]}, \eta_{[0]})$ is invertible. I.e.,

$$\begin{pmatrix} v_{[1]} \\ \lambda_{[1]} \end{pmatrix} = - \begin{pmatrix} J_1^T J_1 & J_2^T \\ J_2 & 0 \end{pmatrix} \begin{pmatrix} -J_1^T \Sigma^{-1} \\ 0 \end{pmatrix} \eta_{[1]} .$$

and thus

$$v_{[1]} = (\mathbf{I}, 0) \begin{pmatrix} J_1^T J_1 & J_2^T \\ J_2 & 0 \end{pmatrix} \begin{pmatrix} J_1^T \Sigma^{-1} \\ 0 \end{pmatrix} \eta_{[1]} .$$

This is the desired result.

□

### 5.2.3. Covariance Matrix and its Computation

In the previous section it has been shown how the solution operator of the least squares estimate can be linearized. The discussion in Chapter 5.1.2 showed that the level of significance, the current estimate and the covariance matrix describe the confidence region to first order. For every sample $\eta(\omega)$ one obtains a different point estimate $\hat{\eta}(\omega)$ of the underlying parameters $\theta \in \Theta$. The parameters $p$ resp. $v$ are derived parameters, i.e., $p = p(\theta)$ and $v = v(\theta)$. Since the estimator $\hat{\eta}$ is a random variable, one can compute the covariance matrix of the random variable $g \circ v \circ \hat{\eta}$ by

$$\begin{aligned} \mathrm{Cov}(g \circ v \circ \hat{\eta}) =& \mathbb{E}[(g \circ v \circ \hat{\eta} - \mathbb{E}[g \circ v \circ \hat{\eta}])(g \circ v \circ \hat{\eta} - \mathbb{E}[g \circ v \circ \hat{\eta}])^T] \\ \approx& K \mathbb{E}[J^+ (\hat{\eta} - \mathbb{E}\hat{\eta})(\hat{\eta} - \mathbb{E}\hat{\eta})^T J^{+T}] K^T \\ =& K J^+ \Sigma^2 J^{+T} K^T \\ =& K P M^{-1} \begin{pmatrix} J_1^T J_1 & 0 \\ 0 & 0 \end{pmatrix} M^{-1} P^T K^T , \end{aligned} \tag{5.29}$$

where

$$K := \frac{\partial g}{\partial v}(v)\Big|_{v=v(\hat{\eta}(\omega))} .$$

The function $g$ is here again a derived parameter as explained in Chapter 5.1. This function can for instance be used to scale parameter values or to model key performance indicators. One defines

$$C := KPM^{-1} \begin{pmatrix} J_1^T J_1 & 0 \\ 0 & 0 \end{pmatrix} M^{-1} P^T K^T \tag{5.30}$$

and calls it the (approximate) *covariance matrix* $C \in \mathbb{R}^{N_g \times N_q}$. Note that two different approximations have been used: Firstly, the linearization of the solution operator and secondly, the Jacobian of the constrained least squares problem $J^+$ is evaluated at the current available sample $\eta(\omega)$, i.e., not at the true value.

In principle one could use (5.30) for the covariance matrix computation. However, as shown by Bock et al. [2007a] it is possible to simplify the expression.

**Lemma 5.2.5.** *Let $C \in \mathbb{R}^{N_g \times N_g}$, $K \in \mathbb{R}^{N_g \times N_v}$, $J_1 \in \mathbb{R}^{N_\eta \times N_v}$ and $J_2 \in \mathbb{R}^{N_{F2} \times N_v}$ be matrices as defined above. If LICQ and PD are satisfied, then the covariance matrix $C$ can be computed by*

$$C = K(\mathbf{1}, 0) \begin{pmatrix} J_1^T J_1 & J_2^T \\ J_2 & 0 \end{pmatrix}^{-1} \begin{pmatrix} \mathbf{1} \\ 0^T \end{pmatrix} K^T \,, \tag{5.31}$$

*where $\mathbf{1} = \mathbf{1}_{N_v} \in \mathbb{R}^{N_v \times N_v}$ is the identity matrix and $0 = 0_{N_v, N_{F2}} \in \mathbb{R}^{N_v \times N_{F2}}$.*

*Proof.* Let $C = K\tilde{C}K^T$ and write $M^{-1} = \begin{pmatrix} X & Y \\ Y^T & Z \end{pmatrix}$. LICQ and PD are sufficient conditions for the nonsingularity of $M$. To compute the block matrices $X, Y, Z$ it is necessary to solve the following linear system:

$$\begin{aligned} \begin{pmatrix} \mathbf{1} & 0 \\ 0 & \mathbf{1} \end{pmatrix} &= \begin{pmatrix} J_1^T J_1 & J_2^T \\ J_2 & 0 \end{pmatrix} \begin{pmatrix} X & Y \\ Y^T & Z \end{pmatrix} \\ \Leftrightarrow \quad \mathbf{1} &= J_1^T J_1 X + J_2^T Y^T \\ 0 &= J_1^T J_1 Y + J_2^T Z \\ 0 &= J_2 X \\ \mathbf{1} &= J_2 Y \,. \end{aligned}$$

From 5.30 the covariance matrix can be computed as follows:

$$\begin{aligned} \tilde{C} =& (\mathbf{1}, 0) \begin{pmatrix} X & Y \\ Y^T & Z \end{pmatrix} \begin{pmatrix} J_1^T J_1 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} X & Y \\ Y^T & Z \end{pmatrix} \begin{pmatrix} \mathbf{1} \\ 0 \end{pmatrix} \\ =& (\mathbf{1}, 0) \begin{pmatrix} X J_1^T J_1 X & X J_1^T J_1 Y \\ Y^T J_1^T J_1 X & Y^T J_1^T J_1 Y \end{pmatrix} \begin{pmatrix} \mathbf{1} \\ 0 \end{pmatrix} \\ =& X J_1^T J_1 X \\ =& X(\mathbf{1} - J_2^T Y^T) \\ =& X - X J_2^T Y \\ =& X \,. \end{aligned}$$

□

**Proposition 5.2.6.** *Let $C \in \mathbb{R}^{N_g \times N_g}$, $K \in \mathbb{R}^{N_g \times N_v}$, $J_1 \in \mathbb{R}^{N_\eta \times N_v}$ and $J_2 \in \mathbb{R}^{N_{F2} \times N_v}$ be matrices as defined above. If LICQ and PD are satisfied then the covariance matrix $C$ can be*

*computed by*

$$C = K Q_2^T \left( Q_2 J_1^T J_1 Q_2^T \right)^{-1} Q_2 K^T \,, \tag{5.32}$$

*where $Q_2$ results from the QR decomposition of $J_2$, i.e.*

$$J_2^T = (Q_1^T, Q_2^T) \begin{pmatrix} L^T \\ 0 \end{pmatrix} .$$

*The shapes of the matrices are as follows: $L \in \mathbb{R}^{N_{F2} \times N_{F2}}$, $Q_1 \in \mathbb{R}^{N_{F2} \times N_v}$, $Q_2 \in \mathbb{R}^{N_v - N_{F2} \times N_v}$, $J_2 \in \mathbb{R}^{N_{F2} \times N_v}$, $J_1 \in \mathbb{R}^{N_\eta \times N_v}$.*

*Proof.* The proof is basically the proof given in Kostina et al. [2009], Körkel [2002]. In the following $C = K\tilde{C}K^T$ and for convenience $C \equiv \tilde{C}$.

$$\begin{pmatrix} J_1^T J_1 & J_2^T \\ J_2 & 0 \end{pmatrix}^{-1} \begin{pmatrix} \mathbf{1} \\ 0 \end{pmatrix} = \begin{pmatrix} C \\ D \end{pmatrix}$$

$$\Leftrightarrow \qquad \mathbf{1} = J_1^T J_1 C + J_2^T D \tag{$*$}$$

$$0 = J_2 C \,. \tag{$**$}$$

Performing a $QR$ decomposition of $J_2^T$ yields

$$J_2^T = (Q_1^T, Q_2^T) \begin{pmatrix} L^T \\ 0 \end{pmatrix} \text{ and therefore } J_2 = L Q_1 \,.$$

Inserting this relation in $(**)$ yields $0 = LQ_1C$. In consequence, one finds that

$$Q_1 C = 0$$

since $L$ is nonsingular due to (LICQ). Looking at Eqn. $(*)$ one can apply the transformation

$$\begin{aligned} \mathbf{1} &= J_1^T J_1 (Q_1^T, Q_2^T) \begin{pmatrix} Q_1 \\ Q_2 \end{pmatrix} C + Q_1^T L^T D \\ &= J_1^T J_1 \left( Q_1^T \underbrace{Q_1 C}_{=0} + Q_2^T Q_2 C \right) + Q_1^T L^T D \,. \end{aligned}$$

Multiplying $Q_2$ from the left, and using $Q_2 Q_1^T = 0$, yields

$$\begin{aligned} Q_2 &= Q_2 \left( J_1^T J_1 \right) Q_2^T Q_2 C \\ \text{and} \quad Q_2 C &= \left( Q_2 J_1^T J_1 Q_2^T \right)^{-1} Q_2 \\ \text{thus} \quad Q_2^T Q_2 C &= (Q_2^T Q_2 + Q_1^T Q_1) C = C \\ C &= Q_2^T \left( Q_2 J_1^T J_1 Q_2^T \right)^{-1} Q_2 \,. \end{aligned}$$

This is the desired result. □

### 5.2.4. Key Performance Indicators

In chemical engineering it is often the case that the exact values of the parameters are not the final objective. Rather, the parameter estimates are used in a subsequent process optimization. Because the numerical estimates of the parameters are uncertain, it follows that also the dy-

namics are uncertain and hence all quantities computed from the solution of the dynamics are not known without error. Quantities of interest are called *key performance indicators* (KPI) in the chemical engineering literature. Consider for instance the ODE

$$\begin{aligned} \dot{y} =& f(t, y, u, p) \\ y(t_0) =& y_0 \ , \end{aligned}$$

where one component $y_i(t)$ of $y(t) \in \mathbb{R}^{N_y}$ is the substance of interest. A KPI could for instance be the amount of substance at time $t_e$. There are two possibilities how the confidence region of key performance can be estimated.

One option is to use the estimated parameters $\hat{p}$ (or correspondingly $\hat{v}$) as inputs of the function $g(\hat{p})$. Using linear error propagation one has already found $J^+$. This matrix can be used as seed matrix for a subsequent directional derivative $\frac{\partial g}{\partial p} \cdot J^+$. Alternatively, it may also be a good idea to pre-accumulate $\frac{\partial g}{\partial p}$ by propagation of $p_{[0]} + \mathbf{I}_{N_p} T$. As these directions need to be propagated anyway to compute $J_1$ and $J_2$, one can compute $K$ rather cheaply in the same integrator call.

Another possibility is to introduce the KPI as an additional (pseudo-)parameter $p_{KPI}$ defined by an implicit equation. In the above example one could use

$$0 = p_{KPI} - y(t_e; y_0, u, p) \ .$$

and set $p := (p, p_{KPI})$. I.e., one has to solve the nonlinear constrained least squares problem

$$\begin{aligned} \min_{p \in \mathbb{R}^{N_p}} & \frac{1}{2} \| F_1(t_{\text{mts}}, y(t_{\text{mts}}; y_0, u, p), p) \|_2^2 \\ \text{s.t.} \quad & 0 = p_{KPI} - y(t_e; y_0, u, p) \\ \text{where} \quad & \dot{y} = f(t, y, u, p) \\ & y(t_0) = y_0 \text{ is satisfied by } \quad y(t; y_0, u, p) \ . \end{aligned}$$

This has the advantage that the objective function does not have to be modified.

# 6. Optimum Experimental Design

In Chapter 5 it has been discussed how parameters $p \in \mathbb{R}^{N_p}$ can be estimated using the least squares objective function. The discussion also involved how the goodness of the estimates can be quantified by confidence regions. It regularly happens that the confidence region is very large, possibly of infinite size in certain directions. The experimental setup, described by the control functions $u(t) \in \mathbb{R}^{N_u}$ and weights $w \in \mathbb{R}^{N_{\text{mts}} \times N_h}$, influences the size of confidence set. Naturally, it is desired to find the optimal experiment that leads to the "smallest" confidence region.

One should note that the proposed controls $u(t)$ and weights $w$ depend on the current estimate of the parameter estimate $\hat{p}(\omega)$. In consequence, the overall approach to find parameter estimates with small confidence region is of iterative nature:

1. Based on the current estimate $\hat{p}(\omega)$ of the parameter $p$ it is possible to find an *optimal experimental design*, described by the control functions $u(t)$ and measurement weights $w$.
2. By use of the proposed controls $u(t)$ and weights $w$ one performs an *experiment*. This yields new measurement samples.
3. The old and the new measurement samples are used in a parameter estimation where one obtains a new parameter estimate $\hat{p}(\omega)$ and a confidence region.
4. If the confidence region is "small" enough, stop, otherwise go to 1.

This meta-algorithm is called *sequential approach*. It is also possible to plan several parallel experiments at once (the so-called *parallel approach*). See Figure 6.2 for a graphical illustration.

For an overview of optimum experimental design, see the textbook Pukelsheim [1993] (optimality theory of experimental designs in linear models) and for the current state of the art the survey papers by Franceschini and Macchietto [2008] and Pronzato [2008].

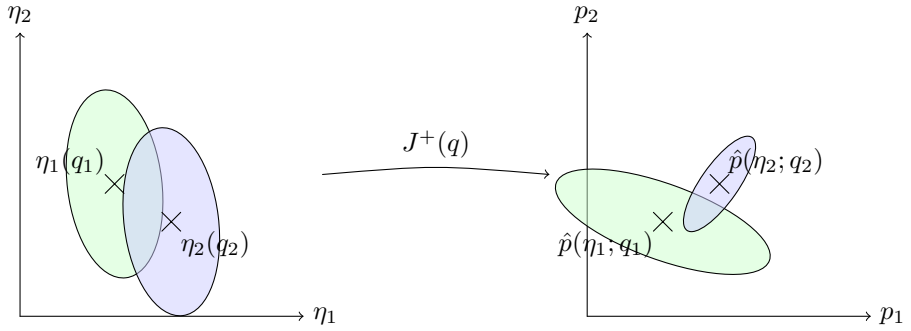



Figure 6.1.: This figure shows two experiments where different controls $q$ are used. The measurements $\eta_1 \equiv \eta_1(q_1)$ and $\eta_2 \equiv \eta(q_2)$ depend on the experimental setup. Their true values are not known, but it is possible to provide confidence regions. Since every measurement is different, even for the same experimental setup, one generally obtains different parameter estimates. To obtain the approximate confidence region of the parameters one can linearize the solution operator $J^+$ (which depends on the controls $q$) of the parameter estimation and apply linear error propagation. Hence, the confidence sets of the parameter estimates may be of different size and shape.

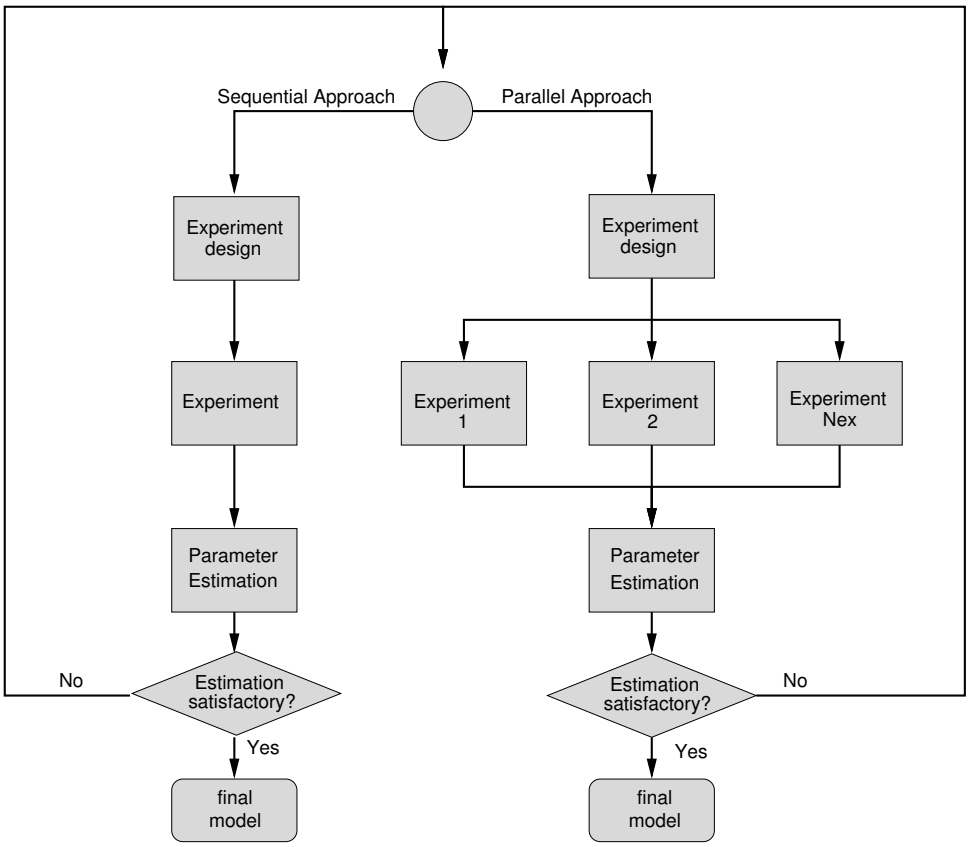



Figure 6.2.: The overall process to obtain parameter estimates with small confidence regions requires the iteration between experimental design optimization, experiment and parameter estimation. On the left, the sequential approach is depicted while on the right one can see the a graphical explanation of the parallel approach.

## 6.1. Nonlinear Program for Experimental Design Optimization

It is necessary to define a function that maps the covariance matrix $C$ to a real number. Such objective functions are denoted $\Phi(C) \in \mathbb{R}$. They are typically statistically motivated and allow a geometric interpretation. See for instance the book by Pukelsheim [1993]. Let $A \in \mathbb{R}^{N\times N}$ be a symmetric positive definite matrix, then equation $y^T Ay = 1$ defines an ellipsoid where the eigenvectors of $A$ are the principal directions of the ellipsoid and the inverse of the square root of the eigenvalues are the corresponding equatorial radii. As shown in the previous section, the random variable $g \circ v \circ \hat{\eta}$ is, to first order, normally distributed, i.e., $g \circ v \circ \hat{\eta} \sim \mathcal{N}(g \circ v \circ \hat{\eta}(\omega), C)$. I.e., one can use (5.5) to construct an (approximate) confidence region by setting $A = C^{-1} \in \mathbb{R}^{N_g\times N_g}$. Using an eigenvalue decomposition one obtains $y^T C^{-1} y = y^T Q^T \operatorname{diag}(1/\lambda_1, \ldots, 1/\lambda_{N_g}) Qy = z^T \operatorname{diag}(1/\lambda_1, \ldots, 1/\lambda_{N_g}) z = \gamma^2_{N_g}(\alpha)$. Hence, if $z = \mu_i e_i$ it follows that $\mu_i = \pm\sqrt{\lambda_i}\gamma_{N_g}(\alpha)$ is the equatorial radius of the confidence ellipsoid. See Lemma 5.1.1 for the notation. A graphical interpretation can be found in Figure 6.3. Popular choices are

1. ***A*-criterion:** The arithmetic mean of the eigenvalues of the covariance matrix:

$$\Phi_A(C) := \frac{1}{N_g} \operatorname{tr}(C) \,, \tag{6.1}$$

where $C \in \mathbb{R}^{N_g\times N_g}$.

2. ***E*-criterion:** The largest eigenvalue of the covariance matrix:

$$\Phi_E(C) := \max(\lambda(C)) \ . \tag{6.2}$$

The scaled square root of the $E$-criterion $\gamma_{N_g}(\alpha)\sqrt{\Phi_E(C)}$ corresponds to the largest principal half-axis of the confidence ellipsoid.

3. ***D*-criterion:** The geometric mean of the eigenvalues of the covariance matrix:

$$\Phi_D(C) := \det(C)^{1/N_g} \ . \tag{6.3}$$

The volume of a enclosing box of the confidence ellipsoid is $2\gamma_{N_g}(\alpha)\sqrt{\Phi_D(C)}$. Note that $\det(C) = 0$ when one eigenvalue vanishes. This situation occurs in constrained parameter estimation problems when some of the additional parameters do not depend on the measurements. One can use another derived parameter $g$ to circumvent this situation.

4. ***M*-criterion:** picks the largest diagonal element of the covariance matrix

$$\Phi_M(C) := \max_{n=1,\dots,N_g} \sqrt{C_{nn}} \ , \tag{6.4}$$

and are called the *alphabetical objective functions*.

One should notice that, since the overall problem is nonlinear, none of the objective functions is scale invariant. The parameters haven often numerical values of very different magnitudes. Consider for instance the following case where the first parameter is $1 \pm 10\%$ and a second parameter $10000 \pm 10\%$. That means that the variance of the second parameter is of much larger absolute value and hence is also of much higher importance during the optimization. To avoid that the second parameter dominates in the optimization, one often scales the parameters to 1. This has the effect that all parameters are equally "important". In the above example one could use the derived parameter $g(v) = (v_1/1, v_2/10000)$ to do the scaling.

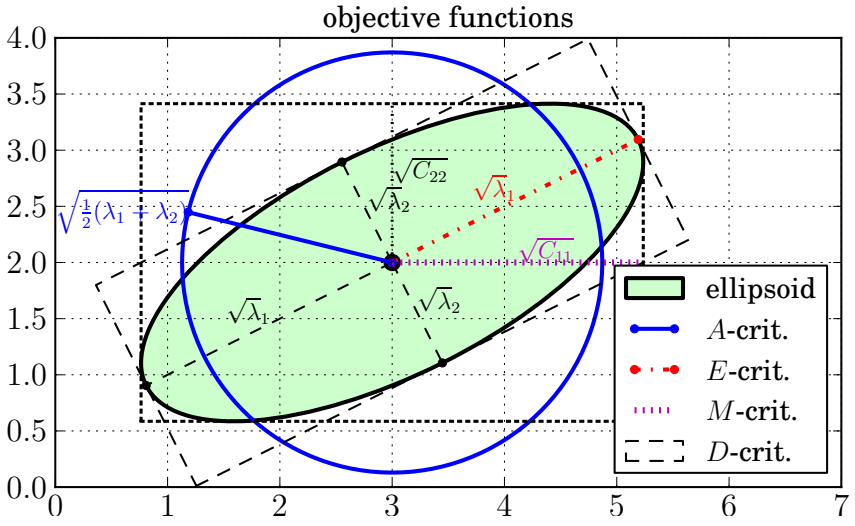



Figure 6.3.: This illustration shows how the OED objective functions are related to the confidence region described by the covariance matrix $C = [[5, 2], [2, 2]]$ and $\gamma^2(\alpha) = 1$. The eigenvalues of the covariance matrix $C$ are $\lambda_1$ and $\lambda_2$.

**Definition 6.1.1** (Mixed Integer Nonlinear Program of a Single Experiment)**.** Let $x(t; x_0, u, p) \in \mathbb{R}^{N_x}$ be the unique, sufficiently smooth, state trajectory defined by the initial values $x_0 \in \mathbb{R}^{N_x}$,

control functions $u \equiv u(t) \in \mathbb{R}^{N_u}$ and parameters $p \in \mathbb{R}^{N_p}$. Let the solution $p_*$ of

$$\begin{aligned} p_*, x_0 = \operatorname{argmin}_{p,x_0} & \frac{1}{2}\|F_1(x(t_{\text{mts}}; x_0, u, p), p; \eta)\|_2^2 \\ \text{s.t.} \quad & 0 = F_2(t_{\text{cstr}}, x(t_{\text{cstr}}; x_0, u, p), p) \\ \text{where} \quad & 0 = M(t, x, u, p) \\ & 0 = x(t_0) - x_0 \quad \text{is satisfied by } x(t; x_0, u, p) \end{aligned} \tag{6.5}$$

be known (c.f. (5.17)). $F_2$ includes boundary value constraints $0 = r(t_{\text{cstr}}, x(t_{\text{cstr}}; x_0, u, p), p)$. Assume that the covariance matrix $C$ is well-defined. The following mixed integer nonlinear program minimizes one measure of the confidence region:

$$\begin{aligned} \min_{x_0 \in \mathbb{R}^{N_x}, u(t) \in \mathbb{R}^{N_u}, w \in \mathbb{R}^{N_{\text{mts}} \times N_h}} & \Phi(C) \\ \text{where} \quad C = & K(\mathbf{1}, 0) \begin{pmatrix} \tilde{J}_1^T W(w)^T \Sigma^{-2} W(w) \tilde{J}_1 & J_2^T \\ J_2 & 0 \end{pmatrix}^{-1} \begin{pmatrix} \mathbf{1} \\ 0^T \end{pmatrix} K^T \\ \tilde{J}_1 = & \frac{\partial}{\partial v} h(x(t_{\text{mts}}; x_0, u, p_*), p_*) \\ J_2 = & \frac{\partial}{\partial v} F_2(t_{\text{cstr}}, x(t_{\text{cstr}}; x_0, u, p_*), p_*) \\ \text{and} \quad 0 = & M(t, x(t), u(t), p_*) \\ 0 = & x(t_0) - x_0 \quad \text{is satisfied by} \quad x(t; x_0, u, p_*) \\ \text{s.t.} \quad 0 = & F_2(t_{\text{cstr}}, x(t_{\text{cstr}}; x_0, u, p_*), p_*) \\ lo \leq & \psi(t, x(t), u(t), p_*) \leq up \\ 0 = & \chi(t, x(t), u(t), p_*) \\ w \in & \mathbb{N}_0^{N_{\text{mts}} \times N_h} \\ w_{\text{total}} \geq & \sum_{i=1}^{N_{\text{mts}}} \sum_{j=1}^{N_h} w_{ij} \ . \end{aligned} \tag{6.6}$$

The generic constraints $\psi$ and $\chi$ collect various constraints such as state and control constraints and $K = \frac{\partial g}{\partial p}$ is the Jacobian of some derived parameter. The weight matrix $W$ depends on the weights $w$ as described in Chapter 5.2.1. The constraints of the parameter estimation problem are also constraints of the experimental design optimization.

## 6.2. Numerical Optimization

There are many possibilities to perform the numerical optimization of the mixed integer nonlinear program in Definition 6.1.1. See for instance the book by Floudas [1995]. The solution of such integer constrained problems is NP hard and therefore often very time-consuming to solve. To obtain acceptable runtimes of the optimization one can relax the integer constraints as described by Körkel [2002]. This relaxation allows one to apply Newton-type optimizers and therefore one can hope for superlinear convergence. As shown in Chapter 6.2.1 the relaxation can be derived relatively easily under some conditions that are generally not a restriction in practice. In Chapter 6.2.3 it is shown how the required derivatives can be computed using algorithmic differentiation.

### 6.2.1. Relaxation of the Integer Constraints

An experimenter is confronted with the question how often, if at all, he should make a measurements of one or more measurement functions. More precisely, at each measurement time $t_{n_{\text{mts}}}$, $n_{\text{mts}} = 1, \ldots, N_{\text{mts}}$, the weight $w_{n_{\text{mts}}} \in \mathbb{N}_0^{N_h}$ defines how many times each component $h_i(t_{n_{\text{mts}}})$, $i = 1, \ldots, N_h$, is measured. See Chapter 5.2.1 for the notation. One would like to relax from $w_{n_{\text{mts}}} \in \mathbb{N}_0^{N_h}$ to $w_{n_{\text{mts}}} \in \mathbb{R}^{N_h}$ to avoid a mixed integer nonlinear program. It is now shown how this can be accomplished.

The weights $w \in \mathbb{N}_0^{N_{\text{mts}} \times N_h}$ specify the number of rows of the matrix $W(w)$. This is a formulation which is impossible to relax and one has to find a more convenient formulation. Let the measurements be statistically independent, i.e., assume that the covariance matrix takes the form

$$\Sigma_{n_{\text{mts}}} := \begin{pmatrix} \sigma_1 \mathbf{I}_{w_{n_{\text{mts}},1}} & & \\ & \ddots & \\ & & \sigma_{N_h} \mathbf{I}_{w_{n_{\text{mts}},N_h}} \end{pmatrix} \in \mathbb{R}^{\sum_{n_h=1}^{N_h} w_{n_{\text{mts}},n_h} \times \sum_{n_h=1}^{N_h} w_{n_{\text{mts}},n_h}}$$

for each measurement time $t_{n_{\text{mts}}}$, where $\mathbf{I}_{w_{n_{\text{mts}},n_h}}$ is the identity matrix. The overall covariance matrix of the measurements is assumed to be diagonal

$$\Sigma := \begin{pmatrix} \Sigma_1 & & \\ & \ddots & \\ & & \Sigma_{N_{\text{mts}}} \end{pmatrix} \in \mathbb{R}^{N_\eta \times N_\eta} .$$

The weights $w$ enter the objective function of the least squares functional through the Jacobian $J_1 := \frac{\partial F_1}{\partial p} = \Sigma^{-1} W(w) \frac{\partial h}{\partial p}$. The matrix-valued function

$$G(w) := J_1^T J_1 = \frac{\partial h}{\partial p}^T W(w)^T \Sigma^{-2} W(w) \frac{\partial h}{\partial p} \in \mathbb{R}^{N_v \times N_v}$$

offers a suitable ansatz for the relaxation since its size is independent of the weights $w$.

**Lemma 6.2.1.** *Let $w \in \mathbb{N}_0^{N_h}$ be weights at some measurement time and $W = W(w) \in \mathbb{N}_0^{M \times N_h}$ the associated weight matrix as defined in (5.12) (hence $M = \sum_{n_h=1}^{N_h} w_{n_h}$). Also, let the diagonal matrix*

$$\Sigma := \begin{pmatrix} \sigma_1 \mathbf{I}_{w_1} & & \\ & \ddots & \\ & & \sigma_{N_h} \mathbf{I}_{w_{N_h}} \end{pmatrix} \in \mathbb{R}^{M \times M}$$

*be given. Then the equation*

$$W^T \Sigma^{-2} W = \operatorname{diag}\left(\frac{w_1}{\sigma_1^2}, \ldots, \frac{w_{N_h}}{\sigma_{N_h}^2}\right)$$

*holds.*

*Proof.*

$$W = \begin{pmatrix} 1_{w_1} & & \\ & \ddots & \\ & & 1_{w_{N_h}} \end{pmatrix}, \; \Sigma^{-2} W = \begin{pmatrix} \frac{1_{w_1}}{\sigma_1^2} & & \\ & \ddots & \\ & & \frac{1_{w_{N_h}}}{\sigma_{N_h}^2} \end{pmatrix}, \; W^T \Sigma^{-2} W = \begin{pmatrix} \frac{1_{w_1}^T 1_{w_1}}{\sigma_1^2} & & & \\ & \frac{1_{w_2}^T 1_{w_2}}{\sigma_2^2} & & \\ & & \frac{1_{w_3}^T 1_{w_3}}{\sigma_3^2} & \\ & & & \ddots \end{pmatrix}$$

where $1_{w_i} = (\underbrace{1, 1, \ldots, 1}_{w_i \text{ times}})^T$, $i = 1, \ldots, N_h$. □

This Lemma is the basis for the following Lemma where all measurement times are taken into account.

**Lemma 6.2.2.** *Let $w \in \mathbb{N}^{N_{\text{mts}} \times N_h}$ and $\sigma \in \mathbb{N}^{N_{\text{mts}} \times N_h}$ be given, where each measurement time corresponds to one row. Then the overall weight matrix is $W = \text{diag}(W_1, \ldots, W_{N_{\text{mts}}}) \in \mathbb{R}^{N_\eta \times N_{\text{mts}} N_h}$ and $\Sigma = \text{diag}(\Sigma_1, \ldots, \Sigma_{N_{\text{mts}}})$. It follows that*

$$W^T \Sigma^{-2} W = \text{diag}(\frac{w_{1,1}}{\sigma_{1,1}^2}, \ldots, \frac{w_{N_{\text{mts}}, N_h}}{\sigma_{\text{mts}, N_h}^2})$$

*holds.*

**Example 6.2.1.** Let the measurement function $h(x(t; x_0, u, p))$ be in $\mathbb{R}^4$, there are three measurement times, i.e., $N_{\text{mts}} = 3$ and the weights/weight matrix and the standard deviation $\sigma$ be

$$w = \begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 3 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix} \quad \Rightarrow \qquad W = \begin{pmatrix} 1 & 0 & 0 & 0 & & & & & & & & \\ 0 & 0 & 0 & 1 & & & & & & & & \\ 0 & 0 & 0 & 1 & & & & & & & & \\ & & & & 0 & 1 & 0 & 0 & & & & \\ & & & & 0 & 1 & 0 & 0 & & & & \\ & & & & 0 & 1 & 0 & 0 & & & & \\ & & & & & & & & 1 & 0 & 0 & 0 \\ & & & & & & & & 0 & 1 & 0 & 0 \\ & & & & & & & & 0 & 0 & 1 & 0 \\ & & & & & & & & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix} \quad \Rightarrow \qquad \Sigma = \text{diag}(1, 4, 4, 6, 6, 6, 9, 10, 11, 12) \, .$$

and hence

$$W^T \Sigma^{-2} W = \text{diag}(\frac{1}{1}, \frac{0}{2^2}, \frac{0}{3^2}, \frac{2}{4^2}, \frac{0}{5^2}, \frac{3}{6^2}, \ldots, \frac{1}{12^2}) \, .$$

### 6.2.2. Nonlinear Program

With the previously discussed relaxations and assumptions it is possible to reformulate the problem into a form suitable for an optimization with a standard gradient based optimizer. I.e.,

- the dynamics are described by a relaxed semi-implicit quasi-linear DAE
- The control functions $u(t) = u(t; q)$ are parametrized by the control vector $q \in \mathbb{R}^{N_q}$ as described in Chapter 4.3
- it is assumed that there is a numerical scheme that can compute the state $x(t; x_0, u(t; q), p)$ given controls $q \in \mathbb{R}^{N_q}$, parameters $p \in \mathbb{R}^{N_p}$ and initial values $x_0 = (y_0, z_0) \in \mathbb{R}^{N_x}$ such that the model equations are satisfied
- the state $x(t; x_0, u(t; q), p)$ is sufficiently smooth in $x_0$, $q$ and $p$
- the initial values $x_0$ are possibly only defined by boundary value constraints.
- $h(x(t_{\text{mts}}; x_0, u(t; q), p), p)$ be a twice continuously differentiable measurement function

- LICQ and PD hold
- The measurements are assumed to be independent, i.e., $\Sigma \in \mathbb{R}^{N_\eta \times N_\eta}$ is diagonal as stated in Lemma 6.2.1.
- the measurement weights are relaxed to the real numbers $w \in \mathbb{R}^{N_{\text{mts}} \times N_h}$

Then the nonlinear program for a single experiment is given by

$$
\begin{aligned}
\min_{x_0 \in \mathbb{R}^{N_x}, q \in \mathbb{R}^{N_q}, w \in \mathbb{R}^{N_{\text{mts}} \times N_h}} \quad & \Phi(C) \\
\text{s.t.} \quad 0 =& F_2(t_{\text{cstr}}, x(t_{\text{cstr}}; x_0, u(t_{\text{cstr}}; q), p_*), x_0, u(t_{\text{cstr}}; q), p_*) \\
lo \leq& \psi(t, x(t; x_0, u(t; q), p_*), u(t; q), p_*) \leq up \quad \forall t \in [t_0, t_e] \\
0 =& \chi(t, x(t; x_0, u(t; q), p_*), u(t; q), p_*) \\
w_{\text{total}} \geq & \sum_{n_{\text{mts}}=1, n_h=1}^{N_{\text{mts}}, N_h} w_{n_{\text{mts}}, n_h} \\
\text{where} \quad C =& K(\mathbf{l}, 0) \begin{pmatrix} J_1^T J_1 & J_2^T \\ J_2 & 0 \end{pmatrix}^{-1} \begin{pmatrix} \mathbf{l} \\ 0^T \end{pmatrix} K^T \\
J_1 =& \operatorname{diag}(\operatorname{vec}(\frac{\sqrt{w}}{\sigma})) \frac{\partial}{\partial v} (h(x(t_{\text{mts}}; x_0, u(t_{\text{mts}}; q), v), p_*)) \in \mathbb{R}^{N_{\text{mts}} N_h \times N_v} \\
J_2 =& \frac{\partial}{\partial v} (F_2(t_{\text{cstr}}, x(t_{\text{cstr}}; x_0, u(t_{\text{cstr}}; q), p_*), x_0, u(t_{\text{cstr}}; q), p_*)) \in \mathbb{R}^{N_{\text{cstr}} N_{F2} \times N_v} \\
\text{and} \quad A(t, y, z, p_*)\dot{y} =& f(t, y, z, u(t; q), p_*) \\
0 =& g(t, y, z, u(t; q), p_*) \\
0 =& y(t_0) - y_0 \\
0 =& z(t_0) - z_0 \quad \text{is satisfied by } x(t; x_0, u, p)
\end{aligned}
\tag{6.7}
$$

The generic constraints $\psi$ and $\chi$ collect various constraints such as state and control constraints and $F_2$ the same constraints as in the constrained least squares problem. The constraint $F_2$ contains potentially also boundary value constraints. That means the initial values $x_0$ change during the optimization and hence $v \equiv (x_0, p)$. The $\sqrt{w}/\sigma$ is taken elementwise. The operation $\operatorname{vec}(A)$ reshapes a multi-dimensional array into a $1-D$ array.

### 6.2.3. Evaluating the OED Objective Function and its Gradient

For the numerical optimization of the NLP in Section 6.2.2 it is necessary to compute the gradient of the objective function w.r.t. the control vector $q \in \mathbb{R}^{N_q}$. The number of controls $N_q$ can grow rapidly, e.g., when they parametrize control functions or when many parallel experimental designs are to be optimized. In contrast, the number of parameters $N_v$ is often relatively small ($N_v \leq 10$) and therefore the derivatives $\frac{\partial h}{\partial v}$ and $\frac{\partial F_2}{\partial v}$ can be evaluated in the forward mode. The overall computational graph is depicted in Figure 6.4. As one can see, the DAE integrator is regarded as a function

$$
(x_0, q, p) \mapsto \begin{pmatrix} x_1, x_2, \ldots, x_{N_{\text{mts}}} \\ x_1, x_2, \ldots, x_{N_{\text{cstr}}} \end{pmatrix} .
$$

I.e., given initial values $x_0$, controls $q$ and parameters $p$, the integrator returns the state at the measurement times $t_{\text{mts}}$ and constraint times $t_{\text{cstr}}$. To compute the Jacobians $\frac{\partial h}{\partial v}$ and $\frac{\partial F_2}{\partial v}$ one can use first-order univariate Taylor polynomial arithmetic as explained in Chapter 2.4. As

initialization one uses

$$\begin{aligned}[q]_2 =& q_{[0]} + 0T\\ [w]_2 =& w_{[0]} + 0T\\ [v]_2 =& v_{[0]} + \mathbf{I}_{N_v} T\ .\end{aligned}$$

The seed matrix (see (2.17)) is the identity matrix $\mathbf{I}_{N_v}$. In Appendix D.4 it is briefly explained how one can differentiate the solver at the example of the explicit and implicit Euler method. Solving the DAE in univariate Taylor arithmetic yields the states at the measurement times

$$\begin{aligned}[x_1]_2 =& x_{1;[0]} + x_{1;[1]} T\\ \vdots =& \vdots\\ [x_{N_{\text{mts}}}]_2 =& x_{N_{\text{mts}};[0]} + x_{N_{\text{mts}};[1]} T\ ,\end{aligned}$$

and the constraint times

$$\begin{aligned}[x_1]_2 =& x_{1;[0]} + x_{1;[1]} T\\ \vdots =& \vdots\\ [x_{N_{\text{cstr}}}]_2 =& x_{N_{\text{cstr}};[0]} + x_{N_{\text{mts}};[1]} T\ .\end{aligned}$$

From these values one can evaluate

$$h_{[0]} + h_{[1]} T = E_2(h)(\begin{pmatrix} [x_0]_2 \\ [x_1]_2 \\ \vdots \\ [x_{N_{\text{mts}}}]_2 \end{pmatrix}, [p]_2)\ ,$$

and

$$F_{2;[0]} + F_{2;[1]} T = E_2(F_2)(t_{\text{cstr}}, \begin{pmatrix} [x_0]_2 \\ [x_1]_2 \\ \vdots \\ [x_{N_{\text{cstr}}}]_2 \end{pmatrix}, [p]_2)\ ,$$

where

$$h_{[1]} \in \mathbb{R}^{N_{\text{mts}} N_h \times N_v} = \frac{\partial h}{\partial v}(x(t_{\text{mts}}; x_0, u, p), p) \tag{6.8}$$

$$F_{2;[1]} \in \mathbb{R}^{N_{\text{cstr}} N_{F2} \times N_v} = \frac{\partial F_2}{\partial v}(t_{\text{cstr}}, x(t_{\text{cstr}}; x_0, u, p), p) \tag{6.9}$$

are the desired derivative. At this point it is necessary to extract certain Taylor coefficients and populate the matrices $J_1 \in \mathbb{R}^{N_{\text{mts}} N_h \times N_v}$ and $J_2 \in \mathbb{R}^{N_{F2} \times N_v}$. It is advantageous to write the extraction as the following matrix product:

$$J_1 = \operatorname{diag}(\operatorname{vec}(\sqrt{w}/\sigma)) \underbrace{(0, \mathbf{I})}_{\in \mathbb{R}^{N_{\text{mts}} N_h \times 2 N_{\text{mts}} N_h}} \underbrace{\begin{pmatrix} h_{1;[0;1]} & h_{1;[0;2]} \cdots & h_{1;[0;N_v]} \\ \vdots & & \vdots \\ h_{N_{\text{mts}};[0;1]} & h_{N_{\text{mts}};[0;2]} \cdots & h_{N_{\text{mts}};[0;N_v]} \\ h_{1;[1;1]} & h_{1;[1;2]} \cdots & h_{1;[1;N_v]} \\ \vdots & & \vdots \\ h_{N_{\text{mts}};[1;1]} & h_{N_{\text{mts}};[1;2]} \cdots & h_{N_{\text{mts}};[1;N_v]} \end{pmatrix}}_{\in \mathbb{R}^{2 N_{\text{mts}} N_h \times N_v}}, \tag{6.10}$$

where for instance $h_{n_{\text{mts};[0;1]}} \in \mathbb{R}^{N_h}$ is the zeroth coefficient and first direction at the $n_{\text{mts}}$th measurement time. Note that in the zeroth coefficient all directions are the same, i.e., $h_{n_{\text{mts};[0;1]}} = \cdots = h_{n_{\text{mts};[0;N_v]}}$. The matrix $J_2$ is similarly computed.

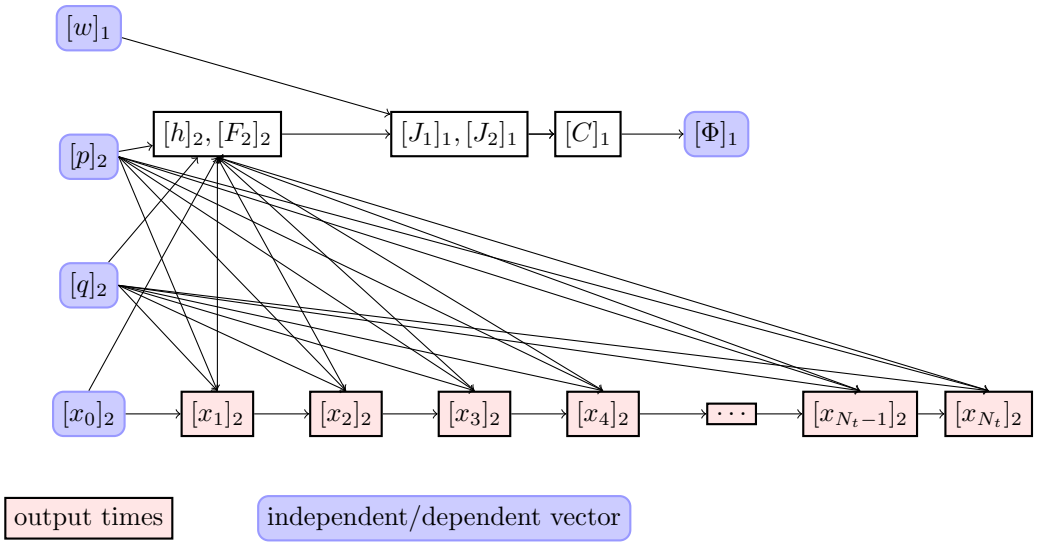



Figure 6.4.: The overall computational graph to evaluate the optimum experimental design objective function: The integrator solves the initial value problem and computes $N_t$ intermediate steps: some of them are either times when the constraints are checked or measurement times. The values at the measurement times are used to to compute the univariate Taylor polynomials $[h]_2$ and $[F_2]_2$. From the Taylor coefficients of $[h]_2$ and $[F_2]_2$ one constructs the Jacobians $[J_1]_1$ and $[J_2]_1$ from which the covariance matrix $[C]_1$ is computed. As final step the objective function maps to the real numbers $\mathbb{R}$.

Once $J_1$ and $J_2$ are known, the covariance matrix can be computed using the result from Proposition 5.2.6. Listing 6.1 shows an implementation in Python. One can see that a QR decomposition is used to find an orthonormal basis of the nullspace of $J_2$ and an additional QR decomposition to avoid the multiplication of two potentially ill-conditioned matrices. The computational graph of the function is shown in Figure 6.5. One can easily add also the objective function evaluation $\Phi(C)$ to the computational graph or even construct more elaborate functions. This computational graph can be used to evaluate the function and it also enables us to apply the reverse mode of AD. I.e., all intermediate values are stored in the computational graph and are therefore available when they are required for the pullback.

```
import algopy; import numpy
from algopy import sum, inv, qr, dot, zeros, solve, sqrt

def eval_C(dhdv, sigma, w, J2):

    NF2, Nv = J2.shape

    # STEP 1: Compute J1 (uses broadcasting, avoids diag matrix)
    J1 = (dhdv.T * (sqrt(w)/sigma).T).T

    # STEP 2: compute Q2, spans the nullspace of J2
    Q,R = algopy.qr_full(J2.T)
    Q2  = Q[:,NF2:].T

    # Step 2: compute C = (J1^T J1)^{-1}
    Q,R = qr(dot(J1,Q2.T))
    Id = numpy.eye(R.shape[0])
    tmp1 = solve(R.T, Id)
```

```
19      C  = solve(R, tmp1)

        return dot(Q2.T, dot(C, Q2))[:Nv,:Nv]
```

Listing 6.1: A Python implementation for the evaluation of the covariance matrix as derived in Proposition (5.2.6). In line 9 one can see that one can avoid the construction of the diagonal matrix and replace it by an elementwise multiplication between tensors of different order. This operation is called broadcasting and is described in Oliphant [2006].

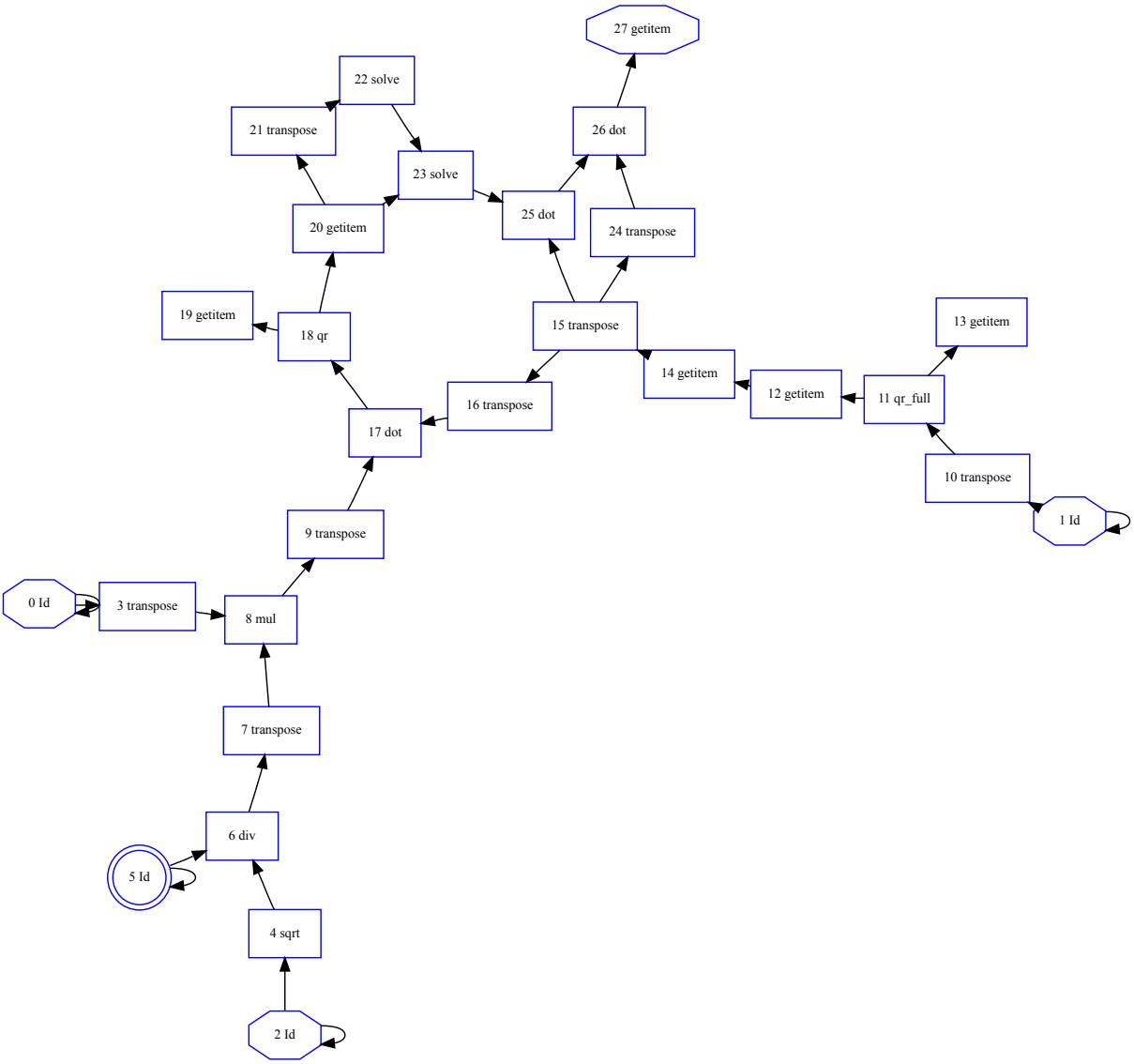



Figure 6.5.: The computational graph of the function shown in Listing 6.1. E.g., the node with id 2 are the weights $w$.

After the objective function $\Phi$ has been evaluated and the intermediate variables haven been stored, it is possible to apply the reverse mode of algorithmic differentiation. I.e., one performs successive pullbacks of the the linear form $\bar{\Phi}\dot{\Phi}$ until one finds

$$\begin{aligned}\bar{\Phi}\dot{\Phi} =& \bar{\Phi}\frac{\partial \Phi}{\partial q}\dot{q} + \bar{\Phi}\frac{\partial \Phi}{\partial p}\dot{p} + \bar{\Phi}\frac{\partial \Phi}{\partial x_0}\dot{x}_0 + \bar{\Phi}\frac{\partial \Phi}{\partial w}\dot{w} \\ =& \bar{q}^T\dot{q} + \bar{p}^T\dot{p} + \bar{x}_0^T\dot{x}_0 + \bar{w}^T\dot{w}\ ,\end{aligned}$$

as explained in Chapter 2.5. With the initialization $\bar{\Phi} = 1$ one therefore obtains the desired gradient $\nabla_q \Phi \equiv \bar{q}^T$. When $x_0$ is a function of $q$ one simply performs another pullback $\bar{x}_0^T \dot{x}_0 = \bar{x}_0^T \frac{\partial x_0}{\partial q} \dot{q}$ and increments $\bar{q}^T := \bar{q}^T + \bar{x}_0^T \frac{\partial x_0}{\partial q}$.

The interesting operation in the reverse mode is the extraction of the Taylor coefficients as described in (6.10). It states that the elements of the Jacobian matrix depend on the first coefficients of a univariate Taylor polynomial, which depend themselves on $q_{[0]}$ and $q_{[1]}$. That means one applies here the result of Lemma 2.5.2, i.e., performs a standard pullback of the mapping from Taylor coefficients to Taylor coefficients.

## 6.3. Parameter Robust Experimental Design Optimization

The experimental design optimization is performed on the current estimate $\hat{p}(\omega)$ of the unknown parameter $p$. This uncertainty may have a negative effect on the sequential approach. Put in formulas, the optimum experimental design objective function reads

$$\xi_* = \text{argmin}_{\xi \in \mathbb{R}^{N_\xi}} \Phi(\xi, \hat{p}(\omega)) \ ,$$

where

$$\xi := (x_0, q, w) \in \mathbb{R}^{N_\xi} \ .$$

There are two possibilities to avoid this dependency: Either by use of minimax or by use of a Bayesian approach where some prior of the true parameters $p$ is assumed. The minimax approach states that one should choose $\xi_*$ such that the objective function is small even if the true parameter $p$ lies in a "difficult" region:

$$\xi_* = \text{argmin}_{\xi \in \mathbb{R}^{N_\xi}} \max_{p \in \text{CR}(\hat{p}(\omega))} \Phi(\xi, p) \ , \tag{6.11}$$

where $\text{CR}(\hat{p}(\omega))$ is a confidence region about the estimate $\hat{p}(\omega)$ of the true parameter value to some confidence level $\alpha$. To avoid the solution of a semi-infinite optimization problem, one can linearize the objective function $\Phi(\xi, p)$ in $p$. The linearity makes it possible to solve the maximization $\max_{p \in \text{CR}(\hat{p}(\omega))}$ analytically. One obtains an approximation of the parameter robust objective function

$$\Phi_{\text{robust}}(\xi) = \Phi(\xi, \hat{p}(\omega)) + \gamma(\alpha) \sqrt{\nabla_p \Phi(\xi, \hat{p}(\omega))^T C^{-1} \nabla_p \Phi(\xi, \hat{p}(\omega))} \ , \tag{6.12}$$

where $C = C(\xi, \hat{p}(\omega))$ is the covariance matrix of the parameter estimation. For the full discussion see the paper by Bock et al. [2007a] or the PhD thesis by Körkel [2002] where also uncertainties in the constraints are treated.

Now to the question how the gradient $\nabla_\xi \Phi_{\text{robust}}(\xi)$ can be computed in terms of univariate Taylor polynomial arithmetic and the reverse mode of AD. The small number of parameters $N_p$ suggests to use the forward mode to compute $\nabla_p \Phi$. That means, to compute the objective function one can proceed as follows:

1. At first, evaluates the Jacobian

$$J_1 = \text{diag}(\frac{\sqrt{w}}{\sigma}) \ \frac{\partial}{\partial T} h(x(t_{\text{mts}}, x_{0;[0]} + x_{0;[1]} T, u, p_{[0]} + p_{[1]} T), p_{[0]} + p_{[1]} T) \Big|_{T=0}$$

   and similarly $J_2$ in the forward mode.

2. To compute $\nabla_p \Phi$ in the forward mode, it is necessary to propagate

$$\frac{\partial \Phi}{\partial T_2} (p_{[0]} + \tilde{p}_{[1]} T_2) \Big|_{T_2=0}$$

with a different set of directions $\tilde{p}_{[1]} = \mathbf{I}_{N_p}$.

3. That means in total one needs to propagate a multivariate Taylor polynomial

$$\begin{aligned} J_{1;[0]} + J_{1;[1]}T_2 = \operatorname{diag}(\frac{\sqrt{w}}{\sigma})\frac{\partial h}{\partial T_1}( \\ x(t_{\text{mts}}, x_{0;[0]} + x_{0;[1]}T_1, u, p_{[0]} + p_{[1]}T_1 + \tilde{p}_{[1]}T_2), p_{[0]} + p_{[1]}T_1 + \tilde{p}_{[1]}T_2)\Big|_{T_1=0} \end{aligned} \tag{6.13}$$

4. Using a polarization identity, e.g., (2.21), one finds that one can compute the coefficients of the Jacobian $[J_1]_2 = J_{1;[0]} + J_{1;[1]}T_2$ by univariate Taylor polynomial arithmetic.

5. The matrices $J_{1,[0]}$, $J_{1,[1]}$ and $J_{2;[0]}$, $J_{2;[1]}$ are then used to evaluate both $\Phi(\xi, \hat{p})$ and $\nabla_p \Phi(\xi, \hat{p})$.

To compute $\nabla_\xi \Phi_{\text{robust}}(\xi)$ one has then to apply pullbacks of the linear form defined in Lemma 2.5.2. Working with interpolation/polarization identities means that there are many operations using coefficients of Taylor polynomials.

## 6.4. Planning Additional and Parallel Experiments

There are many situations when a single experiment does not provide enough information to estimate parameters accurately. Important examples are:

- Certain experimental conditions are fixed and cannot be changed. For instance, it may be the case that it is only possible to perform an experiment at a fixed temperature. Thus, some parameters may be not identifiable and it is therefore necessary to perform several experiments at different temperatures. In such a case, one can use the parallel approach (see Figure 6.2). For each experiment, the experimenter can adjust the controls $u(t; q) \in \mathbb{R}^{N_u}$ and the weights $w \in \mathbb{N}_0^{N_{\text{mts}} \times N_h}$. The experiments are indexed by square brackets. E.g.,
$$u[n_{\text{ex}}](t; q[n_{\text{ex}}]) \in \mathbb{R}^{N_u[N_{\text{ex}}]} \text{ for } n_{\text{ex}} = 1, \ldots, N_{\text{ex}}$$
are the control functions of the $n_{\text{ex}}$-th experiment and $N_{\text{ex}}$ is the total number of experiments.

- In the sequential approach, one interleaves an experimental design optimization step with a parameter estimation step. I.e., for all experimental design optimizations, with exception of the first, it is necessary to take information of past experiments into account. Naturally, past experiments cannot be changed anymore and are therefore called *fixed experiments*. The number of fixed experiments is denoted ${N_{\text{ex}}}^{\text{fix}}$.

When a mixed parallel/sequential approach is taken, the overall matrices $J_1 = \frac{\partial F_1}{\partial v}$ and $J_2 = \frac{\partial F_2}{\partial v}$ are given as

$$\left(\frac{\bar{J}_1}{J_1}\right) = \left(\begin{array}{ccccccc} J_{1p}[1] & J_{1x_0}[1] & & & & & \\ J_{1p}[2] & & J_{1x_0}[2] & & & & \\ J_{1p}[3] & & & J_{1x_0}[3] & & & \\ \vdots & & & & \ddots & & \\ \hline J_{1p}[N_{\text{ex}}^{\text{fix}}+1] & & & & J_{1x_0}[N_{\text{ex}}^{\text{fix}}+1] & & \\ \vdots & & & & & \ddots & \\ J_{1p}[N_{\text{ex}}] & & & & & & J_{1x_0}[N_{\text{ex}}] \end{array}\right)$$

$$\left(\frac{\bar{J}_2}{J_2}\right) = \left(\begin{array}{ccccccc} J_{2p}[1] & J_{2x_0}[1] & & & & & \\ J_{2p}[2] & & J_{2x_0}[2] & & & & \\ J_{2p}[3] & & & J_{2x_0}[3] & & & \\ \vdots & & & & \ddots & & \\ \hline J_{2p}[N_{\text{ex}}^{\text{fix}}+1] & & & & J_{2x_0}[N_{\text{ex}}^{\text{fix}}+1] & & \\ \vdots & & & & & \ddots & \\ J_{2p}[N_{\text{ex}}] & & & & & & J_{2x_0}[N_{\text{ex}}] \end{array}\right),$$

where the blocks have the dimensions $J_{1p}[n_{\text{ex}}] \in \mathbb{R}^{N_\eta[n_{\text{ex}}] \times N_p}$ and $J_{1x_0}[n_{\text{ex}}] \in \mathbb{R}^{N_\eta[n_{\text{ex}}] \times N_x}$. $N_\eta$ is the number of measurements in experiment $n_{\text{ex}}$, $N_{\text{ex}}$ is the number of experiment. Since these matrices are typically of very moderate size and the time for their factorizations and multiplications is negligible compared to the time for the simulation of the dynamical system, it is possible to use the formula in Proposition 5.2.6 without further structure exploitation. One should note that the computation is significantly easier when $x_0$ is not implicitly defined by boundary values. New measurements are then appended row-wise. This may be useful for real-time optimization where new data is immediately used to estimate parameters as well as update the experimental design based on the newly obtained parameter estimate.

# 7. Numerical Results

At some point one has to depart from theoretical considerations and implement the theory in software. Naturally, such software should be able to solve relevant problems. The purpose of this Chapter is to demonstrate that the results of this thesis can be applied to problems of industrial relevance.

The BDF integrator DAESOL-II is used for the time integration. It is capable of univariate Taylor polynomial arithmetic and the reverse mode and can be found in the integrator suite SolvIND written by Albersmeyer and Kirches [2007–]. The algorithm MMA (method of moving asymptotes) from the optimization library NLOPT by Johnson is used for the numerical optimization. It is based on conservative convex separable approximations (CCSA) and has been introduced by Svanberg [1987, 2002].

All numerical experiments have been performed on a Dell Latidude D530 with an Intel(R) Core(TM)2 Duo CPU T7300 @2.00GHz with 2048628 kB physical memory on Linux 2.6.32-24-generic. All sources have been compiled with gcc 4.4.3 using the optimization flag -O3. The used software versions are as follows:

- NumPy 1.3
- SciPy 0.7
- AlgoPy 0.3
- PyAdolc 04-February-2011
- PySolvIND 04-February-2011
- EasyOdoe 04-February-2011
- ADOL-C 2.1
- SolvIND revision 1334.

## 7.1. Validation via a Simple System with Known Solution

To validate the correctness of software, it is a good idea to consider examples for which analytical solutions are known. It is then possible to compare the algorithmically computed solution with the analytical solution. This is the purpose of this section.

Consider the initial value problem

$$\begin{aligned} \dot{x} =& p + qx \\ x(0) =& 1 \end{aligned}$$

with explicit solution

$$x(t) = e^{qt} + (e^{qt} - 1)\frac{p}{q} .$$

Using $h(x, u, p) = x(t)$ one obtains

$$\begin{aligned}\frac{\mathrm{d}h}{\mathrm{d}p} =& (e^{qt} - 1)/q \\ J_{n_{\mathrm{mts}}} =& (e^{qt_{n_{\mathrm{mts}}}} - 1)/q \\ M =& \sum_{n_{\mathrm{mts}}=1}^{N_{\mathrm{mts}}} J_{n_{\mathrm{mts}}}^2 \\ C =& \frac{1}{\sum_{n_{\mathrm{mts}}=1}^{N_{\mathrm{mts}}} J_{n_{\mathrm{mts}}}^2} \ .\end{aligned}$$

Using symbolic differentiation results in the expressions

$$\begin{aligned}\frac{\mathrm{d}C}{\mathrm{d}J_k} =& -\frac{2J_k}{(\sum_{n_{\mathrm{mts}}} J_{n_{\mathrm{mts}}}^2)^2} \\ \frac{\mathrm{d}C}{\mathrm{d}q} =& \frac{2q\sum_{n_{\mathrm{mts}}=1}^{N_{\mathrm{mts}}}(e^{qt_{n_{\mathrm{mts}}}} - 1)^2 - q^2(\sum_{n_{\mathrm{mts}}=1}^{N_{\mathrm{mts}}} 2t_{n_{\mathrm{mts}}} e^{qt_{n_{\mathrm{mts}}}}(e^{qt_{n_{\mathrm{mts}}}} - 1)}{(\sum_{n_{\mathrm{mts}}=1}^{N_{\mathrm{mts}}}(e^{qt_{n_{\mathrm{mts}}}} - 1)^2)^2}\end{aligned}$$

Since the covariance matrix $C$ is a $\mathbb{R}^{1\times 1}$ matrix it coincides with the objective function $\Phi$. The used method is a forward integration in univariate Taylor polynomial arithmetic to compute the Jacobian and the reverse mode of algorithmic differentiation to compute the gradient $\nabla_q \Phi_q(q)$. The errors

$$\left|\frac{\Phi_{\mathrm{symbolic}}(q) - \Phi(q)}{\Phi_{\mathrm{symbolic}}(q)}\right| \quad \text{and} \quad \frac{\|\nabla_q \Phi_{\mathrm{symbolic}}(q) - \nabla_q \Phi(q)\|}{\|\nabla_q \Phi_{\mathrm{symbolic}}(q)\|}$$

depend on the relative tolerance used in the integration. In Figure 7.1 one can see how the errors behave.

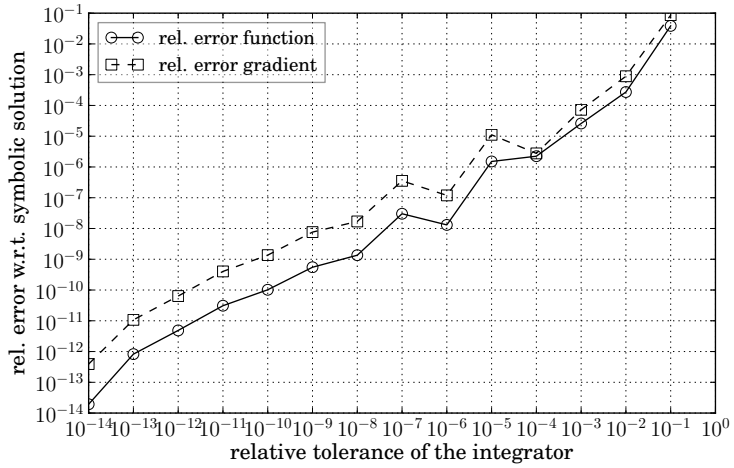



Figure 7.1.: This Figure shows the error between the algorithmically and symbolically computed solution. One can see that the errors are close to the relative tolerance of the integration.

## 7.2. Diels-Alder Reaction

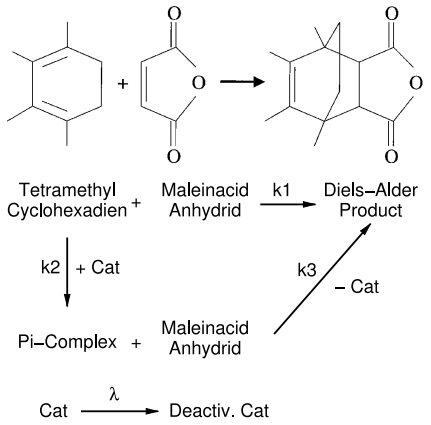



Figure 7.2.: A graphical representation of the reaction paths of the bimolecular catalysis of section 7.2.

This example is also often called the *vpbimolkat* example. It describes a batch reaction of Tetramethyl Cyclohexadien $(y_1, A)$ with Maleinacid Anhydrid $(y_2, B)$ to the Diels-Alder product $(y_3, C)$ in a solvent $(y_4, L)$. There are two possible reaction paths: a direct and a catalyzed. The catalyst degrades over time:

$$\begin{aligned} A + B &\to C \\ K &\xrightarrow{\lambda} K' \end{aligned}$$

A graphical representation is depicted in Figure 7.2. The dynamics are described by the initial value problem

$$\begin{aligned} \dot{y}_1 &= -k \cdot \frac{y_1 \cdot y_2}{m_{tot}}, & y_1(0) &= q_{y_{a1}} \\ \dot{y}_2 &= -k \cdot \frac{y_1 \cdot y_2}{m_{tot}}, & y_2(0) &= q_{y_{a2}} \\ \dot{y}_3 &= \quad k \cdot \frac{y_1 \cdot y_2}{m_{tot}}, & y_3(0) &= 0 \ , \end{aligned}$$

where

$$\begin{aligned} k =& p_{k_1} \exp\left(-\frac{p_{E_1}}{R}\left(\frac{1}{u_T(t)} - \frac{1}{T_{ref}}\right)\right) + p_{k_{kat}} q_{c_{kat}} \exp\left(-p_\lambda t\right) \exp\left(-\frac{p_{E_{kat}}}{R}\left(\frac{1}{u_T(t)} - \frac{1}{T_{ref}}\right)\right) \\ y_4 =& q_{y_{a4}} \\ m_{\text{tot}} =& y_1 M_1 + y_2 M_2 + y_3 M_3 + y_4 M_4 \ . \end{aligned}$$

One can measure the product mass concentration

$$h(y, z, p) = \frac{y_3(t) M_3}{y_1(t) M_1 + y_2(t) M_2 + y_3(t) M_3 + q_{ya4} M_4} 100$$

in which case the standard deviation is $\sigma = 1$.

The controls

$$u(t) = (q_{y_{a1}}, q_{y_{a2}}, q_{y_{a4}}, q_{ckat}, u_T(t)) \in \mathbb{R}^5$$

are the educt molar numbers $q_{ya1}, q_{ya2}, q_{ya4}$, the catalyst concentration $q_{c_{kat}}$ and the temperature profile $u_T(t)$. Upper and lower bounds are $l_q = (0.4, 0.4, 0.4, 0)$ and $u_q = (9, 9, 9, 6)$.

There are five unknown model parameters

$$\begin{aligned} p =& (p_{k_{\text{cat}}}, p_{E_1}, p_{k1}, p_{E_{\text{cat}}}, p_\lambda)^T \in \mathbb{R}^5 \\ \approx& (10^2, 6 \cdot 10^4, 10^{-1}, 4 \cdot 10^4, 0.25) \ . \end{aligned}$$

Two parallel experiments are planned. The constant controls are for the first experiment

$$(q_{y_{a1}}, q_{y_{a2}}, q_{y_{a4}}, q_{ckat}) = (6, 3, 0.6, 0.01)$$

and for the second

$$(q_{y_{a1}}, q_{y_{a2}}, q_{y_{a4}}, q_{ckat}) = (8, 6, 0.8, 0.01) \ .$$

In both experiments there are 20 (potential) measurement times, each with weight $w_{n_{\text{mts}}} = 1$. The control function $u_T(t)$ is piecewise linear continuous in both cases.

There are two constraints on the controls:

$$\begin{aligned} 0.1 \leq& q_{y_{a1}} M_1 + q_{y_{a2}} M_2 + q_{y_{a4}} M_4 \leq 10 \\ 10 \leq& \frac{q_{y_{a1}} M_1 + q_{y_{a2}} M_2}{q_{y_{a1}} M_1 + q_{y_{a2}} M_2 + q_{y_{a4}} M_4} \leq 70 \ . \end{aligned}$$

**Before Optimization** The parameters are scaled to $p = (1, 1, 1, 1, 1)$ for an easier interpretation. I.e., $C_{\text{scaled}} := VCV$, where $V = \text{diag}(\frac{1}{p_1}, \ldots, \frac{1}{p_{N_p}})$. Before optimization one obtains

$$C_{\text{scaled}} = \begin{pmatrix} 5.258e-02 & -1.126e-02 & -3.244e-01 & 8.302e-02 & 1.461e+00 \\ -1.126e-02 & 4.810e-03 & 1.478e-01 & -2.096e-01 & -1.818e-01 \\ -3.244e-01 & 1.478e-01 & 5.298e+00 & -6.964e+00 & -3.760e+00 \\ 8.302e-02 & -2.096e-01 & -6.964e+00 & 1.701e+01 & -7.199e+00 \\ 1.461e+00 & -1.818e-01 & -3.760e+00 & -7.199e+00 & 5.036e+01 \end{pmatrix} .$$

The standard deviations are

$$\begin{pmatrix} 2.293e-01 & 6.936e-02 & 2.302e+00 & 4.124e+00 & 7.097e+00 \end{pmatrix} .$$

They are the square roots of the diagonal of the scaled covariance matrix.

**After the Optimization** The optimization with the $A$-criterion suggests to use

$$(q_{y_{a1}}, q_{y_{a2}}, q_{y_{a4}}, q_{ckat}) = \begin{pmatrix} 7.886e+00 & 7.945e+00 & 4.000e-01 & 1.224e+00 \end{pmatrix}$$

for the first experiment and

$$(q_{y_{a1}}, q_{y_{a2}}, q_{y_{a4}}, q_{ckat}) = \begin{pmatrix} 8.401e+00 & 9.000e+00 & 4.000e-01 & 0.000e+00 \end{pmatrix}$$

for the second.

The covariance matrix is

$$C_{\text{scaled}} = \begin{pmatrix} 6.680e-04 & -1.784e-04 & -7.663e-05 & 3.753e-05 & 1.345e-04 \\ -1.784e-04 & 9.165e-05 & 3.293e-05 & -3.270e-05 & -3.240e-06 \\ -7.663e-05 & 3.293e-05 & 1.042e-03 & -7.165e-04 & -6.739e-05 \\ 3.753e-05 & -3.270e-05 & -7.165e-04 & 8.188e-04 & 2.201e-03 \\ 1.345e-04 & -3.240e-06 & -6.739e-05 & 2.201e-03 & 1.911e-02 \end{pmatrix} .$$

and the standard deviations are

$$\Big(2.585e-02, 9.573e-03, 3.227e-02, 2.861e-02, 1.383e-01 \quad \Big)$$

As one can see in Figure 7.3 the optimization suggests to take measurements only a couple of times.

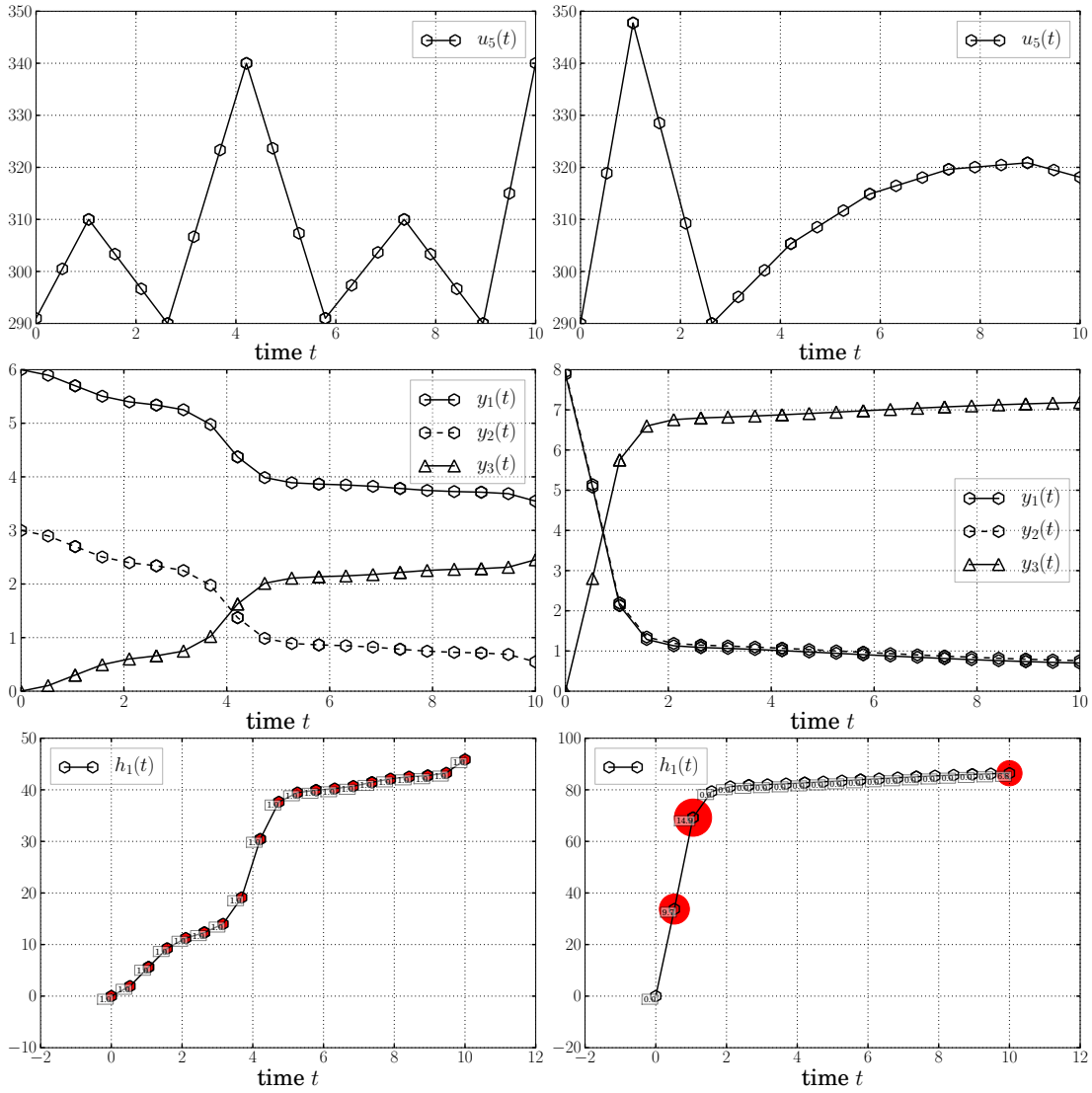



Figure 7.3.: On the left is the initial and on the right the optimized experimental design of the first experiment as described in Chapter 7.2. One can see the measurement function $h(x(t; x_0, u(t), p), p)$ and the measurement weights (plotted as red dots). The larger the dot, the more measurements should be taken. The weights of the initial experimental design are equally distributed and 1 and after optimization most are zero but a few which have weights between 10 and 14.

**Verification of the Computed Objective Function and Gradient** The evaluation of the objective function has been verified against VPLAN written by Körkel [2002] where it was found that the computed Jacobian $J_1$ and covariance matrix $C$ coincided very well.

Based on the verified objective function evaluation is is possible to check whether the gra-

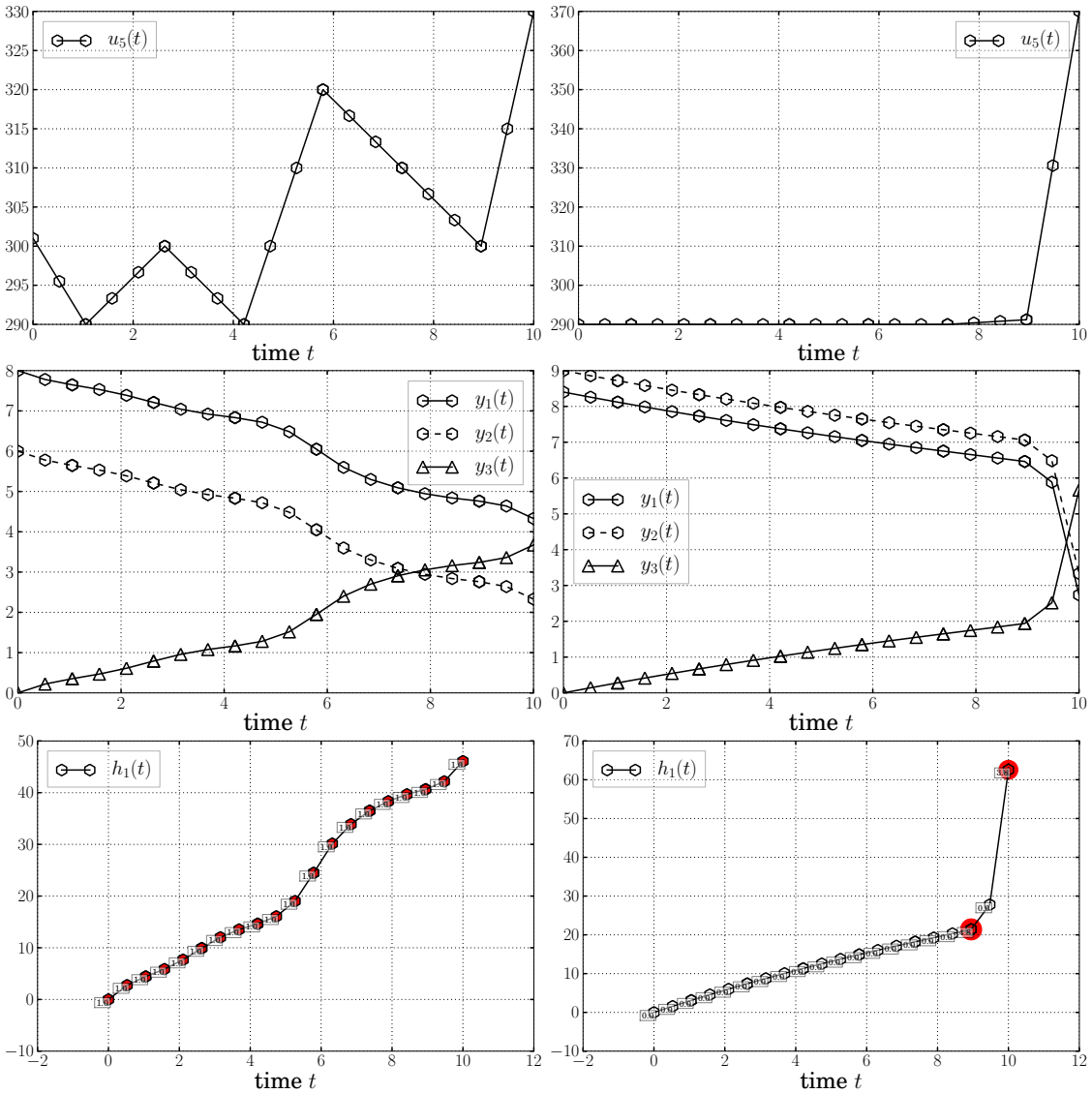



Figure 7.4.: In this Figure shows the dynamics and measurement function of the initial design (left) and the optimized design (right) of second experiment (Chapter 7.2).

dient is approximately correct by use of external differentiation based on a finite differences approximation as explained in Chapter 2.7. Roughly speaking, the finite differences solution

$$\nabla_{\xi;FD}\Phi(\xi) \approx \frac{\Phi(\xi + \epsilon e_i) - \Phi(\xi)}{\epsilon} \text{ for } i = 1, \ldots, N_\xi \tag{7.1}$$

can be up to 8 digits correct for the right choice of $\epsilon$. For the check the rule of thumb $\epsilon q_i \approx 10^{-8}$ has been used. The relative tolerance of the integrator was set to $10^{-4}$. For a single experiment one finds that the element-wise relative difference $\frac{\|\nabla_{\xi;FD}\Phi(\xi) - \nabla_\xi\Phi(\xi)\|}{\|\nabla_\xi\Phi\|}$ is between $10^{-5}$ and $10^{-9}$.

## 7.3. Optimization with Many Controls and Weights

It is not only of interest that the objective function $\Phi(\xi)$ and its gradient $\nabla_\xi \Phi(\xi)$ can be computed with accuracy close to the machine precision but also the runtime is of central importance. While the number of parameters $N_p$ is typically rather small, i.e., of order 10, the number of controls $N_q$ and weights $N_w$ can grow rapidly. In particular, when there are many possible measurement points, or when a fine control function discretization is desired, then $N_\xi = N_q + N_w$ can go into the hundreds or thousands. Using finite differences or the forward mode of AD would mean that the gradient may be up to $N_\xi$ times as expensive as the objective function itself. When the simulation takes already more than a minute it may simply be too expensive to evaluate the gradient.

One should note that due to the structure of the objective function one can speed up the forward mode of AD:

1. The measurement weights $w$ enter the computational graph only after the numerical integration. Since the integration is typically by far the most expensive part of the computation, it is relatively cheap to compute many directional derivatives w.r.t. the measurement weights $w$.

2. With each additional parallel experiment the number of controls increases. However, since the experiments are mutually independent, the number of directions does not grow when the number of experiments is increased.

The advantage of the reverse mode comes into play when there are complex nonlinear interactions. Nonetheless, it is of course possible to check the theoretical prediction that the gradient is only a small multiple as expensive as the function itself. The model from Chapter 7.2, with four parallel experiments, is used for the numerical test. The control function discretization is successively refined. One finds that the gradient can indeed by computed in a small constant multiple of the time to evaluate the function itself, see Figure 7.5.

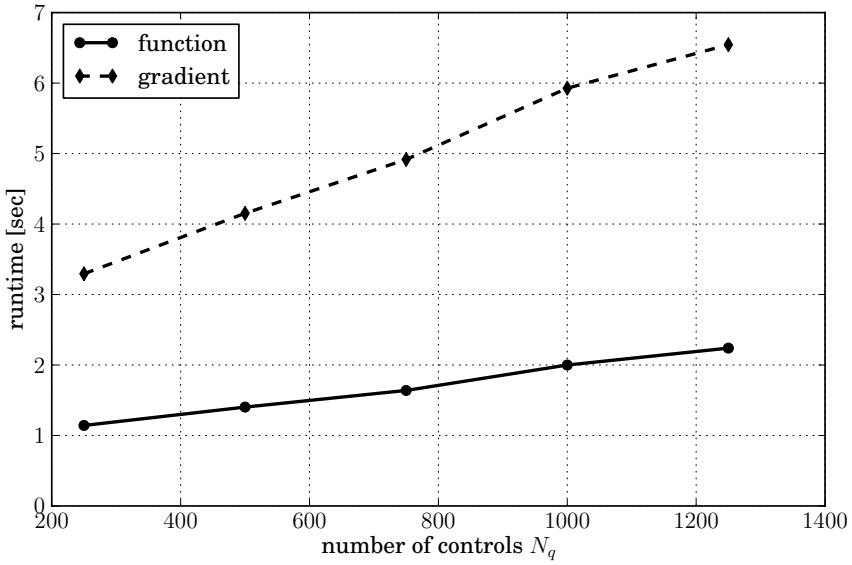



Figure 7.5.: Comparison between the time to compute the objective function and its gradient. One can see that the gradient is about three times as expensive as the function.

For $N_\xi = 816$, with $N_q = 416$ and $N_w = 400$ the result for the first experiment is depicted in Figure 7.6. It shows the temperature, state and measurement function trajectory. The other three experiments are not shown. They contained large, instantaneous jumps in the temperature

profile. Most likely one would have to add some constraints on the rate the temperature can change.

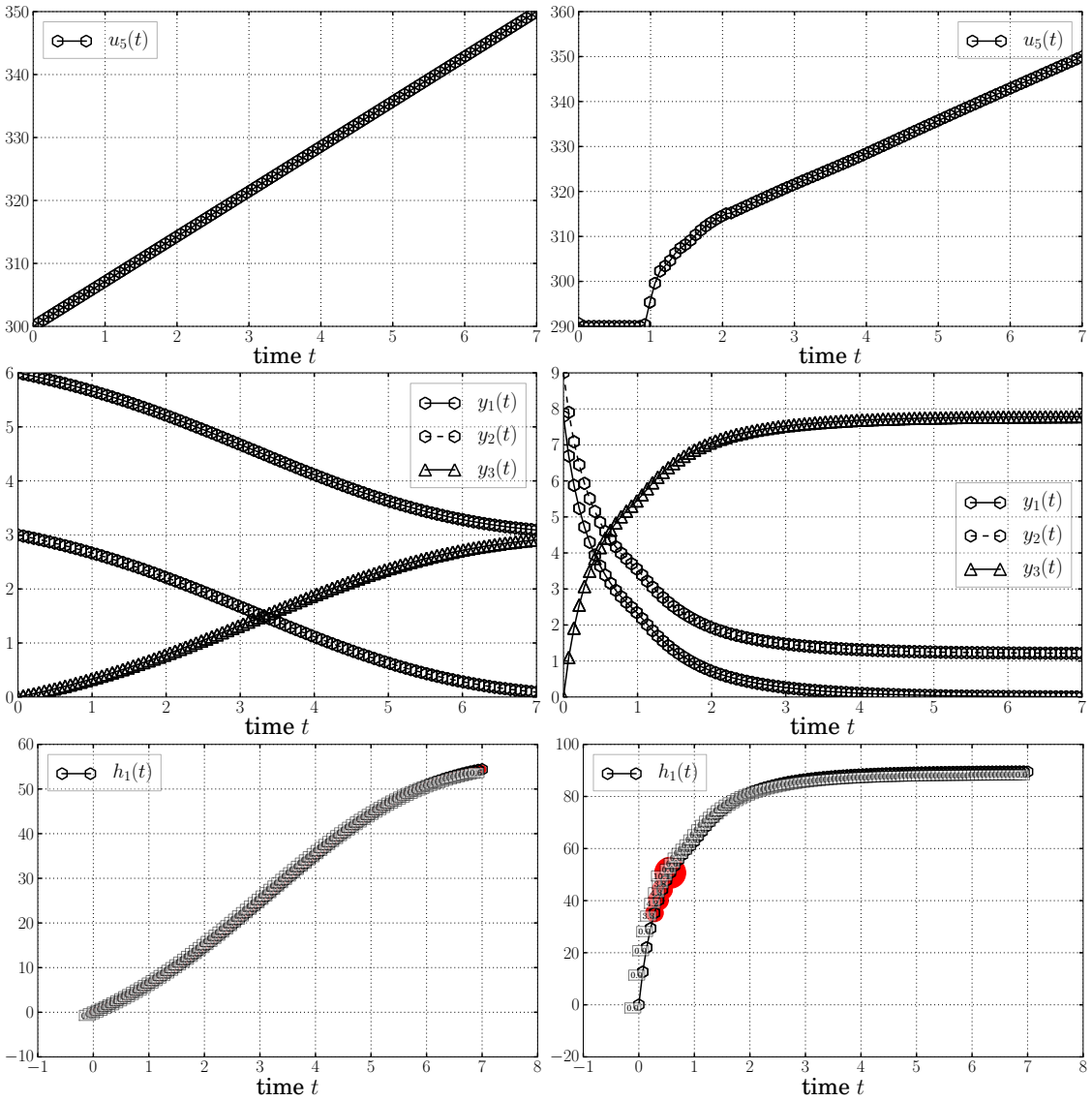



Figure 7.6.: Temperature profile, state and measurement function of the Diels-Alder-reaction. On the left hand side the initial design and on the right hand side the optimized experimental design is shown. See the discussion in Chapter 7.3.

## 7.4. Urethane Reaction

The synthesis of polyurethane is of high-relevance, for instance to produce foams and other synthetic materials. As argued in [Körkel, 2002, Section 10.2] one can gain insights into this process by considering first the simpler synthesis of urethane. It is the goal to produce $C$ (urethane) from the two educts $A$ (phenylisocyanate) and $B$(butanol). From the chemical reaction

$$\begin{aligned} A + B &\rightarrow C \\ A + C &\rightleftharpoons D \\ 3A &\rightarrow E \end{aligned}$$

one can see that also undesired products $D$ (allophanate) and $E$(isocyanurate) are produced. Not shown is the solvent $L$ (dimethylsulfoxide). For implementation reasons, the aliases $x_7 := n_A$, $x_8 := n_B$, $x_1 := n_C$, $x_2 := n_D$, $x_3 = n_E$, $x_9 := n_L$ are used because in this way differential and algebraic variables are separated.

The experimental setup is a semi-batch reactor, i.e., the reaction starts with some initial values $x_{ia}$ $(1 \leq i \leq 6)$ of the substances and there are additional feeds. More precisely, there are two feeds $f_1(t)$ and $f_2(t)$, where $f_1(t)$ adds more of substance $A$ and $L$ and $f_2(t)$ more of substance $B$ and $L$. The amount of substance that has been added up to time $t$ is denoted by $x_{if}(t)$ $(1 \leq i \leq 6$ ) and additionally the notation $x_4 := f_1(t)$, $x_5 := f_2(t)$, $x_6 := T(t)$ is used.

The model is a semi-explicit DAE described by

$$\begin{aligned}
\dot{x}_1 &= V(r_1 - r_2 + r_3) && \text{urethane } (C) \\
\dot{x}_2 &= V(r_2 - r_3) && \text{allophanate } (D) \\
\dot{x}_3 &= V r_4 && \text{isocyanurate } (E) \\
\dot{x}_4 &= u_{r_{f_1}} && \text{feed rate 1} \\
\dot{x}_5 &= u_{r_{f_2}} && \text{feed rate 2} \\
\dot{x}_6 &= u_{r_T} && \text{temperature rate} \\
0 &= x_7 + x_1 + 2x_2 + 3x_3 - q_{x_{7a}} - x_{7f}(t) && \text{phenylisocyanate } (A) \\
0 &= x_8 + x_1 + x_2 - q_{x_{8a}} - x_{8f}(t) && \text{butanol } (B) \\
0 &= x_9 - q_{x_{9a}} - x_{9f}(t) && \text{dimethylsulfoxide } (L) \\
x(0) &= (0, 0, 0, 0, 0, q_{T_0}) \,, && \text{initial values}
\end{aligned}$$

where

$$
\begin{aligned}
V(t) &= \sum_{i=1}^{3} \frac{x_i M_{x_i}}{\rho_{x_i}} + \sum_{i=7}^{9} \frac{x_i M_{x_i}}{\rho_{x_i}} && \text{net volume}\\
M(t) &= \sum_{i=1}^{3} x_i M_{x_i} + \sum_{i=7}^{9} x_i M_{x_i} && \text{net mass}\\
r_1(t) &= k_1 \frac{x_7 x_8}{V^2} && \text{reaction rates}\\
r_2(t) &= k_2 \frac{x_7 x_1}{V^2}\\
r_3(t) &= k_3 \frac{x_2}{V}\\
r_4(t) &= k_4 \frac{x_7^2}{V^2}\\
k_1(t) &= p_{k_{\text{ref1}}} \exp\left(-\frac{p_{E_{a1}}}{R}\left(\frac{1}{x_6} - \frac{1}{T_{\text{ref1}}}\right)\right) && \text{reaction rate constants}\\
k_2(t) &= p_{k_{\text{ref2}}} \exp\left(-\frac{p_{E_{a2}}}{R}\left(\frac{1}{x_6} - \frac{1}{T_{\text{ref2}}}\right)\right)\\
k_3(t) &= k_2/k_c\\
k_4(t) &= p_{k_{\text{ref4}}} \exp\left(-\frac{p_{E_{a4}}}{R}\left(\frac{1}{x_6} - \frac{1}{T_{\text{ref4}}}\right)\right)\\
k_c(t) &= k_{c2} \exp\left(-\frac{p_{\Delta H_2}}{R}\left(\frac{1}{x_6} - \frac{1}{T_{c2}}\right)\right)\\
x_{7f}(t) &= q_{x_{7f_1}} x_4 && \text{amount of substance 7 added by feeds}\\
x_{8f}(t) &= q_{x_{8f_2}} x_5 && \text{amount of substance 8 added by feeds}\\
x_{9f}(t) &= q_{x_{9f_1}} x_4 + q_{x_{9f_2}} x_5 && \text{amount of substance 9 added by feeds .}
\end{aligned}
$$

That means, in total there are nine state variables $x_i(t)$ $1 \leq i \leq 9$. As **control functions** one uses the control variables $q_{x_{7a}}, q_{x_{8a}}, q_{x_{9a}}, q_{x_{7f_1}}, q_{x_{9f_1}}, q_{x_{8f_2}}, q_{x_{9f_2}}$, where $x_{ia}$ are the inital values of the educts and $q_{x_{9f_1}}$ is the relative amount of substance $x_9$ that gets added by feed $f_1$. Furthermore, there are time dependent control functions $u_{r_{f_1}}(t), u_{r_{f_2}}(t)$ and $u_{r_T}(t)$ which model the feed rates and the temperature rate. Combined in a vector valued function this reads

$$u(t) := (q_{x_{7a}}, q_{x_{8a}}, q_{x_{9a}}, q_{x_{7f_1}}, q_{x_{9f_1}}, q_{x_{8f_2}}, q_{x_{9f_2}}, q_{T_0}, u_{r_{f_1}}(t), u_{r_{f_2}}(t), u_{r_T}(t)) \ .$$

**Model parameters** are the activation energies $p_{E_{a1}}, p_{E_{a2}}, p_{E_{a4}}$, the frequency factors $p_{k_{\text{ref1}}}, p_{k_{\text{ref2}}}, p_{k_{\text{ref4}}}$ and the equilibrium constant $p_{K_{c2}}$. For the experimental design optimization, the initial estimates

$$
\begin{aligned}
p_{E_{a1}} &= 35240.0\, J/mol\\
p_{E_{a2}} &= 85000.0\, J/mol\\
p_{E_{a4}} &= 35000.0\, J/mol\\
p_{k_{\text{ref1}}} &= 5.0 \cdot 10^{-4}\, m^3/(h \cdot mol)\\
p_{k_{\text{ref2}}} &= 8.0 \cdot 10^{-8}\, m^3/(h \cdot mol)\\
p_{k_{\text{ref4}}} &= 1.0 \cdot 10^{-8}\, m^3/(h \cdot mol)\\
p_{\Delta H_2} &= -\,17031\, J/mol\\
p_{K_{c2}} &= 0.17\, m^3/mol
\end{aligned}
$$

| molar mass | density | reference temperature |
|---|---|---|
| $M_{x_7} = 0.11911\,kg/mol$ | $\rho_{x_7} = 1095.0\,kg/m^3$ | $T_{\text{ref}1} = 363.16\,K$ |
| $M_{x_8} = 0.07412\,kg/mol$ | $\rho_{x_8} = 809.0\,kg/m^3$ | $T_{\text{ref}2} = 363.16\,K$ |
| $M_{x_1} = 0.19323\,kg/mol$ | $\rho_{x_1} = 1415.0\,kg/m^3$ | $T_{\text{ref}4} = 363.16\,K$ |
| $M_{x_2} = 0.31234\,kg/mol$ | $\rho_{x_2} = 1528.0\,kg/m^3$ | $T_{C2} = 363.16\,K$ |
| $M_{x_3} = 0.35733\,kg/mol$ | $\rho_{x_3} = 1451.0\,kg/m^3$ | gas constant |
| $M_{x_9} = 0.07806\,kg/mol$ | $\rho_{x_9} = 1101.0\,kg/m^3$ | $R = 8.314\,J/(K \cdot mol)$ |

Table 7.1.: Constants in the urethane model.

can be used. Additionally, there are fixed **model constants** which are collected in Table 7.1.

There are three possible types of measurements modeled by the **measurement functions**

$$\begin{aligned} h_1(t,x,u,p) =& \frac{x_7 M_{x_7}}{M(t)} \pm 0.005 \\ h_2(t,x,u,p) =& \begin{bmatrix} \frac{x_1 M_1}{M(t)} \\ \frac{x_2 M_2}{M(t)} \end{bmatrix} \pm \begin{bmatrix} 5 \times 10^{-3} \\ 5 \times 10^{-5} \end{bmatrix} \\ h_3(t,x,u,p) =& \frac{x_3 M_3}{M(t)} \pm 5 \times 10^{-6}\,. \end{aligned}$$

By $\pm 0.005$ is is meant that a measurement has a standard deviation of 0.005.

To stay in a region that is physically meaningful the constant controls have to be non-negative and additionally there are the following **bound constraints**:

$$\begin{aligned} 0.1 \le MV_1 \le 10 \\ 0 \le MV_2 \le 10^4 \\ 0 \le MV_3 \le 100 \\ 0 \le g_1 \le 0.8 \\ 0 \le g_{1f_1} \le 0.9 \\ 0 \le g_{1f_2} \le 10 \\ 0 \le V_a \le 7.5 \times 10^{-4} \end{aligned}$$

$$\begin{aligned} MV_1 &:= \frac{q_{x_{8a}} + q_{x_{8f_2}}}{q_{x_{7a}} + x_{7f_1}} \\ MV_2 &:= \frac{q_{x_{7f_1}}}{q_{x_{7a}}} \\ MV_3 &:= \frac{q_{x_{8f_2}}}{q_{x_{8a}}} \\ g_1 &:= \frac{q_{x_{8a}} \cdot M_8 + q_{x_{7a}} \cdot M_7}{q_{x_{8a}} \cdot M_8 + q_{x_{7a}} \cdot M_7 + q_{x_{9a}} \cdot M_9} \\ g_{1f_1} &:= \frac{x_{8f_1} \cdot M_8}{x_{8f_1} \cdot M_8 + q_{x_{9f_1}} \cdot M_9} \\ g_{1f_2} &:= \frac{x_{7f_2} \cdot M_7}{x_{7f_2} \cdot M_7 + q_{x_{9f_2}} \cdot M_9} \\ V_a &:= \frac{q_{x_{8a}} \cdot M_8}{\rho_8} + \frac{q_{x_{7a}} \cdot M_7}{\rho_7} + \frac{q_{x_{9a}} \cdot M_9}{\rho_9}\,. \end{aligned}$$

**Analysis of the Numerical Optimization** The overall optimization required 936.17 seconds, i.e., about 15 minutes.

| evaluation part | runtime [sec] |
|---|---|
| forward integration | 420.901785 |
| forward measurement function evaluation | 0.210486 |
| forward objective function evaluation with AlgoPy | 0.438100 |
| reverse integration | 514.318237 |
| reverse measurement function evaluation | included in reverse integration |
| reverse objective function evaluation with AlgoPy | 1.424430 |
| overall | 936.174747 |

Table 7.2.: This table shows timings of the numerical optimization performed on a system as described in the beginning of this chapter. One should note that for one evaluation of the gradient there is one forward integration and one reverse integration.

### Optimization Results

Two parallel experiments are planned. The controls are chosen to be

$$(q_{x_{7a}}, q_{x_{8a}}, q_{x_{9a}}, q_{x_{7f_1}}, q_{x_{9f_1}}, q_{x_{8f_2}}, q_{x_{9f_2}}) = (0.106, 0.106, 0.0876, 0.0319, 0.0319, 0.0486, 0.0454)$$

for the first experiment and

$$(q_{x_{7a}}, q_{x_{8a}}, q_{x_{9a}}, q_{x_{7f_1}}, q_{x_{9f_1}}, q_{x_{8f_2}}, q_{x_{9f_2}}) = (0.206, 0.206, 0.1876, 0.0219, 0.0519, 0.0286, 0.0154)$$

for the second experiment. Per experiment, seven possible measurement times are specified with a total number of 32 measurements. Before optimization the scaled covariance matrix is

$$C_{\text{scaled}} = \begin{pmatrix} 6.066e-03 & 1.929e-03 & 7.378e-03 & 8.354e-03 & 6.214e-03 & 7.987e-03 & 4.126e+00 & 1.447e+00 \\ 1.929e-03 & 7.098e-04 & 2.349e-03 & 2.695e-03 & 2.193e-03 & 2.579e-03 & 1.189e+00 & 4.083e-01 \\ 7.378e-03 & 2.349e-03 & 9.009e-03 & 1.024e-02 & 7.627e-03 & 9.834e-03 & 5.095e+00 & 1.784e+00 \\ 8.354e-03 & 2.695e-03 & 1.024e-02 & 1.303e-02 & 9.787e-03 & 1.254e-02 & 7.123e+00 & 2.460e+00 \\ 6.214e-03 & 2.193e-03 & 7.627e-03 & 9.787e-03 & 7.770e-03 & 9.427e-03 & 4.573e+00 & 1.579e+00 \\ 7.987e-03 & 2.579e-03 & 9.834e-03 & 1.254e-02 & 9.427e-03 & 1.212e-02 & 6.885e+00 & 2.376e+00 \\ 4.126e+00 & 1.189e+00 & 5.095e+00 & 7.123e+00 & 4.573e+00 & 6.885e+00 & 1.200e+04 & 3.782e+03 \\ 1.447e+00 & 4.083e-01 & 1.784e+00 & 2.460e+00 & 1.579e+00 & 2.376e+00 & 3.782e+03 & 1.211e+03 \end{pmatrix}$$

where the standard deviations of the parameters, i.e., the square root of diagonal elements of $C_{\text{scaled}}$ are

$$\Big(7.78e-02,\ 2.66e-02,\ 9.49e-02,\ 1.14e-01,\ 8.81e-02,\ 1.10e-01,\ 1.09e+02,\ 3.48e+01\Big)$$

The optimized experimental design ($A$-criterion) suggests to use

$$(q_{x_{7a}}, q_{x_{8a}}, q_{x_{9a}}, q_{x_{7f_1}}, q_{x_{9f_1}}, q_{x_{8f_2}}, q_{x_{9f_2}}) =$$
$$\Big(1.34e-01 \quad 7.96e-02 \quad 7.00e-02 \quad 6.84e-02 \quad 1.16e-02 \quad 4.62e-02 \quad 6.72e-03\Big)$$

for the first experiment and

$$(q_{x_{7a}}, q_{x_{8a}}, q_{x_{9a}}, q_{x_{7f_1}}, q_{x_{9f_1}}, q_{x_{8f_2}}, q_{x_{9f_2}}) =$$
$$\Big(2.70e-01 \quad 1.84e-01 \quad 0.00e+00 \quad 6.28e-02 \quad 0.00e+00 \quad 3.54e-02 \quad 0.00e+00\Big)$$

for the second. The weights do not change much, possibly an artifact of the optimizer. In particular, one would expect the weight at the initial time $t_0$ to be zero, since the derivative of measurement function w.r.t. $p$ vanishes. That is, in the first measurement there is no information at all. A finite differences check shows that the gradient w.r.t. the weights are correct up to square root of the machine precision. Also, one can see that though the weights have not changed a lot, the initial measurement time has the lowest weight and probably it would go down to zero when the optimizer is allowed many more iterations. Possibly, with a better optimizer a better solution could be found. Despite this fact, a very good reduction in the objective function can be observed: The scaled covariance matrix, i.e., when all parameter are scaled to 1, is

$$C_{\text{scaled}} = \begin{pmatrix} 1.24e-05 & 1.55e-07 & 2.41e-07 & 8.01e-06 & 8.11e-09 & 9.81e-08 & -1.10e-05 & -3.07e-07 \\ 1.55e-07 & 4.55e-08 & -1.58e-10 & 7.45e-08 & -2.18e-08 & -6.26e-09 & -4.35e-07 & -2.93e-07 \\ 2.41e-07 & -1.58e-10 & 1.90e-08 & 2.49e-07 & 5.36e-09 & 1.63e-08 & -1.89e-07 & 5.73e-08 \\ 8.01e-06 & 7.45e-08 & 2.49e-07 & 1.05e-05 & -1.45e-08 & 2.07e-07 & -5.79e-06 & 3.58e-06 \\ 8.11e-09 & -2.18e-08 & 5.36e-09 & -1.45e-08 & 2.29e-08 & 7.69e-09 & -3.53e-09 & -5.59e-08 \\ 9.81e-08 & -6.26e-09 & 1.63e-08 & 2.07e-07 & 7.69e-09 & 1.77e-08 & -2.90e-08 & 1.14e-07 \\ -1.10e-05 & -4.35e-07 & -1.89e-07 & -5.79e-06 & -3.53e-09 & -2.90e-08 & 2.49e-05 & 1.92e-05 \\ -3.07e-07 & -2.93e-07 & 5.73e-08 & 3.58e-06 & -5.59e-08 & 1.14e-07 & 1.92e-05 & 2.49e-05 \end{pmatrix}$$

and the scaled standard deviations of the parameters, i.e., the square root of diagonal elements of $C_{\text{scaled}}$ are

$$\Big(3.52e-03,\ 2.13e-04,\ 1.37e-04,\ 3.24e-03,\ 1.51e-04,\ 1.33e-04,\ 4.99e-03,\ 4.99e-03\Big)$$

That means the confidence region of the parameters has been reduced significantly.

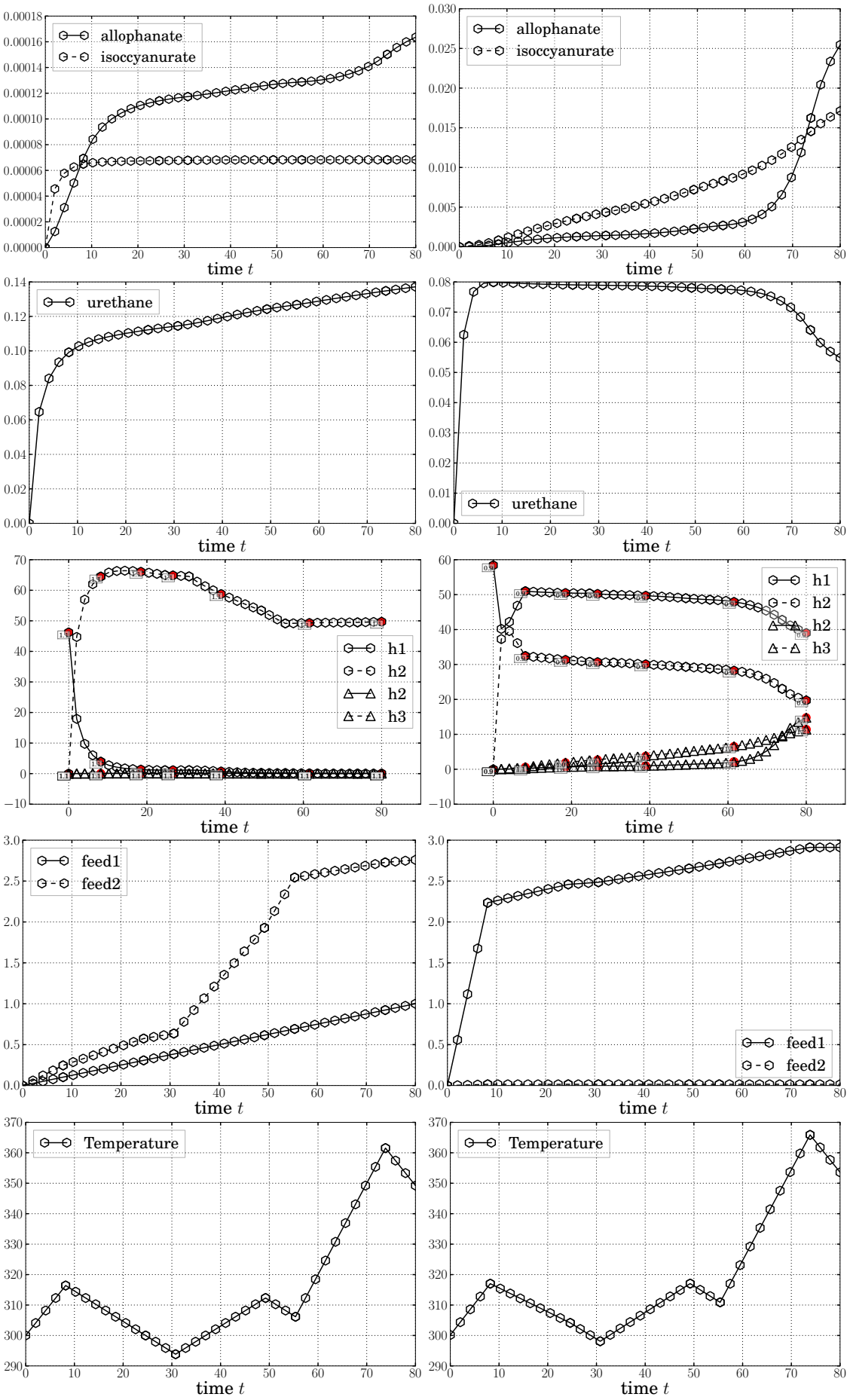



Figure 7.7.: On the left side the first initial experiment is shown and on the right the simulation of the optimized experimental design (See discussion in Chapter 7.4).

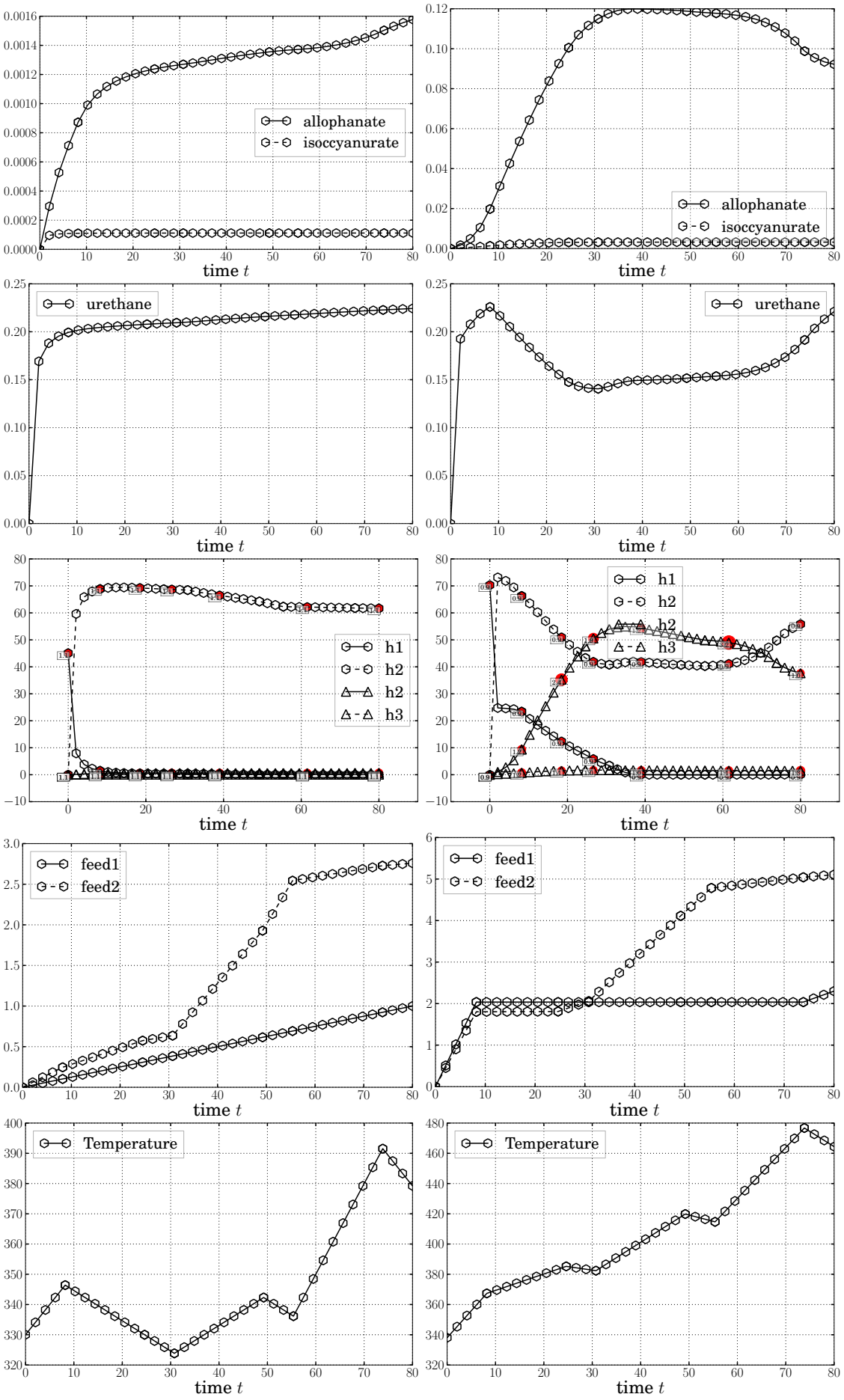



Figure 7.8.: On the left side the second initial experiment is shown and on the right the simulation of the optimized experimental design (See discussion in Chapter 7.4).

# 8. Conclusions

## 8.1. Summary

The theory of algorithmic differentiation and univariate Taylor polynomial arithmetic has been generalized to the important class of matrix operations and factorizations. The underlying idea was to use the defining equations of the matrix operations and derive structure exploiting algorithms. It can be seen as a combination of the vast literature on first-order matrix calculus results and the AD theory. The novel algorithms make it possible to reduce the size of the computational graph as well as the memory requirement during the reverse mode of AD. As a proof of concept, the algorithms have been implemented in the software AlgoPy [Walter, 2009] where it was found that the approach works reliably. That the chosen approach is in principle suitable to do high-performance computations has been investigated in Chapter 3.11.

The algorithms were then successfully applied in experimental design optimization of reference examples from chemical engineering. This required the combination of several existing software tools (ADOL-C, SolvIND/DAESOL-II) in the optimum experimental design prototype EasyOdoe. It was found that the optimization of the experimental design significantly reduces the number of measurements required for a reliable parameter estimate. The results have been verified against the software VPLAN. The convergence behavior also indicates that the computed derivatives are correctly evaluated. For the Diels-Alder reaction, it has been demonstrated that the gradient of the OED objective function can be computed in about a factor three of the runtime of the objective function.

## 8.2. Outlook

Hence, with the theory described in this thesis, it will be possible to optimize experiments with with many ($\gg 1000$) controls and weights. Additionally, since algorithms have been derived on the level of numerical linear algebra functions, it will be possible to introduce new, more elaborate, objective functions. Beyond the direct consequences that follow from this thesis, the approach raises also several questions:

1. There are many other useful dense matrix operations and factorizations. A dedicated treatment could be an asset. E.g., Golub and Pereyra [1973] derives formulas for the Fréchet derivative of the Moore-Penrose pseudoinverse. One could generalize this to univariate Taylor polynomial arithmetic and state an algorithm for the reverse mode.

2. Performing parameter estimation with PDE constraints can lead to problems where direct numerical algebra routines are not the method of choice. One could investigate how the algorithms in this thesis generalize to iterative solution processes, for instance based on the discussion in [Bock et al., 2007b].

3. Often, matrix factorizations have to be updated, E.g., when an additional row is appended to a matrix. In that case, it is not clear, whether it pays off to derive structure exploiting algorithms based on the defining equations.

4. Matrices in large-scale programming are typically sparse. It is not clear how well the approach shown in this thesis can be generalized to sparse matrix factorizations.

5. The reference implementation in AlgoPy could be implemented more efficiently to allow high-performance computations. A first step in this direction has already been made in the software TaylorPoly [Walter, 2010]. It is based on the programming language C and strives to be a library of useful algorithms rather than an all-at-once tool like CasADi [Andersson et al., 2010]. The idea is to have a library of tested algorithms similarly to LAPACK which can be used differentiate algorithms by hand, e.g., to aid the design of a differentiated DAE solver in the forward and reverse mode of AD, or to be used in other AD tools. Since everything is based on C without internal memory management, it is typically rather easy to call these functions from other programming languages. Additionally, clean C code is easier to process with compiler tools as the source-to-source compiler ROSE [Quinlan et al., 2011] or Tapenade [Hascoët and Pascual, 2004]. As part of such software one could also add more algorithms. For instance, one could differentiate the Householder QR decomposition and add certain sparse matrix operations.

6. To determine the tractability index of differential algebraic equations one requires in the process evaluations of pivoted QR decompositions of rank-deficient matrices. One could generalize the shown results to treat this case.

7. Several polarization and interpolation techniques have been discussed. They are a valuable tool to compute derivatives by using univariate Taylor polynomial arithmetic. It would be useful to have a general interpolation framework that also allows the efficient evaluation of mixed partial derivatives of the form

$$\left.\frac{\partial^{d+1} f}{\partial T_1^d \partial T_2}(x + v_1 T_1 + v_2 T_2)\right|_{T_1=T_2=0} \quad \text{for } d = 0, \ldots, D-1 \ .$$

8. Altman [2010] reported that multivariate Taylor polynomial arithmetic seems to produce more accurate results than univariate Taylor polynomial arithmetic in combination with interpolation. It would be useful to have an error analysis of the interpolation process.

9. It has been demonstrated that Newton-Hensel lifting can be used to derive an asymptotically fast algorithm for the scalar division $z = x/y$. It may be useful to investigate whether this result can be generalized to the univariate Taylor polynomial arithmetic on the level of matrix operations.

# A. UTP Arithmetic of Hyperbolic and Inverse Trigonometric Functions

The most popular algorithms for univariate Taylor polynomial arithmetic have been described in Chapter 2.3.6. However, the shown algorithms sometimes do not suffice. Most importantly, a treatment of hyperbolic and inverse trigonometric functions is missing. Neidinger [2005] briefly explains the solution approach and Annamalai [2010] derived explicit recurrences. The purpose of this section is to collect these findings and serve as a reference.

**Lemma A.0.1.** *Let $x, y, z : \mathbb{R} \to \mathbb{R}$ be entire functions satisfying the differential equation*

$$\frac{\partial y}{\partial t}(t) = z(t)\frac{\partial x}{\partial t}(t) \ .$$

*Then the coefficients $y_{[d]}$, $d = 1, 2, \ldots$ of the series expansion $y(t) = \sum_{d=0}^{\infty} y_{[d]} t^d$ can be computed from the coefficients $x_{[d]}$ and $z_{[d]}$, $d = 0, 1, \ldots$ using the recurrence*

$$\tilde{y}_{[d]} = \sum_{k=1}^{d} \tilde{x}_{[k]} z_{[d-k]} \ , d = 1, 2, \ldots \tag{A.1}$$

*where $\tilde{y}_{[k]} := k y_{[k]}$.*

*Proof.* The series expansions are $\frac{\partial y}{\partial t}(t) = \sum_{d=0}^{\infty}(d+1) y_{[d+1]} t^d$, $\frac{\partial x}{\partial t}(t) = \sum_{d=0}^{\infty}(d+1) x_{[d+1]} t^d$ and $z(t) = \sum_{d=0}^{\infty} z_{[d]} t^d$. Hence, $\sum_{d=0}^{\infty} \tilde{y}_{[d+1]} t^d = \sum_{d=0}^{\infty} \sum_{k=0}^{d} \tilde{x}_{[k+1]} z_{[d-k]}$. It follows $\tilde{y}_{[d+1]} = \sum_{k=0}^{d} \tilde{x}_{[k+1]} z_{[d-k]}$ and hence $\tilde{y}_{[d]} = \sum_{k=1}^{d} \tilde{x}_{[k]} z_{[d-k]}$. □

**Lemma A.0.2.** *Let $x, y, z : \mathbb{R} \to \mathbb{R}$ be entire functions satisfying the differential equation*

$$z(t)\frac{\partial y}{\partial t}(t) = \frac{\partial x}{\partial t}(t) \ .$$

*Then the coefficients $y_{[d]}$, $d = 1, 2, \ldots$ of the series expansion $y(t) = \sum_{d=0}^{\infty} y_{[d]} t^d$ can be computed from the coefficients $x_{[d]}$ and $z_{[d]}$, $d = 0, 1, \ldots$ using the recurrence*

$$\tilde{y}_d = \frac{1}{z_{[0]}} \left( \tilde{x}_{[d]} - \sum_{k=1}^{d-1} z_{[d-k]} \tilde{y}_{[k]} \right) \ , d = 1, 2, \ldots \tag{A.2}$$

*where $\tilde{y}_{[k]} := k y_{[k]}$.*

*Proof.* Inserting the series expansion into the differential equation one obtains

$$z(t)\frac{\partial y}{\partial t} = \sum_{d=0}^{\infty} \left( \sum_{k=0}^{d} (k+1) y_{[k+1]} z_{[d-k]} \right) t^d = \sum_{d=0}^{\infty} (d+1) x_{[d+1]} t^d \ ,$$

i.e. $\sum_{k=0}^{d} \tilde{y}_{[k+1]} z_{[d-k]} = \tilde{x}_{[d+1]}$. Let $y_{[0]}, \ldots, y_{[d]}$ be already known, then one can separate the yet unknown coefficient $y_{[d+1]}$ as follows: $\sum_{k=0}^{d-1} \tilde{y}_{[k+1]} z_{[d-k]} + \tilde{y}_{[d+1]} z_{[0]} = \tilde{x}_{[d+1]}$. One obtains

$$\tilde{y}_{[d+1]} = \frac{1}{z_{[0]}} \left( \tilde{x}_{[d+1]} - \sum_{k=0}^{d-1} z_{[d-k]} \tilde{y}_{[k+1]} \right)$$

and hence the result follows after substitution of $d+1 \mapsto d$. □

**Lemma A.0.3** (Exponential in UTP arithmetic)**.** *Let $[x]_D \in \mathbb{R}[T]/(T^D)$ be given, then*

$$\begin{aligned} y_{[0]} =& \exp(x_{[0]}) \\ \tilde{y}_{[d]} =& \sum_{k=1}^{d} y_{[d-k]} \tilde{x}_{[k]} \qquad d = 1, \ldots, D-1 \ , \end{aligned}$$

*is a recurrence for*

$$[y]_D = E_D(\exp)([x]_D) \ .$$

*The notation $\tilde{y}_{[k]} := k y_{[k]}$ is used.*

*Proof.* Differentiation of $y(t) = \exp(x(t))$ w.r.t. $t$ yields the differential equation $\dot{y} = \exp(x(t))\dot{x} = y\dot{x}$. Thee result follows from Lemma A.0.1. □

**Lemma A.0.4** (Natural Logarithm in UTP arithmetic)**.** *Let $[x]_D \in \mathbb{R}[T]/(T^D)$ be given, then*

$$\begin{aligned} y_{[0]} =& \ln(x_{[0]}) \\ \tilde{y}_{[d]} =& \frac{1}{x_{[0]}} \left( \tilde{x}_{[d]} - \sum_{k=1}^{d-1} x_{[d-k]} \tilde{y}_{[k]} \right) \qquad d = 1, \ldots, D-1 \ , \end{aligned}$$

*is a recurrence for*

$$[y]_D = E_D(\ln)([x]_D) \ ,$$

*where* ln *is the natural logarithm to the base $e = \exp(1)$. The notation $\tilde{y}_{[k]} := k y_{[k]}$ is used.*

*Proof.* $\dot{y} = \frac{\partial \ln(x(t))}{\partial t} = \frac{\dot{x}}{x}$ and hence $\dot{y}x = \dot{x}$. The result then follows from Lemma A.0.2. □

**Lemma A.0.5** (Sine and Cosine in UTP arithmetic)**.** *Let $[x]_D \in \mathbb{R}[T]/(T^D)$ be given, then*

$$\begin{aligned} s_{[0]} =& \sin(x_{[0]}) \\ c_{[0]} =& \cos(x_{[0]}) \\ \tilde{s}_{[d]} =& \sum_{k=1}^{d} \tilde{x}_{[k]} c_{[d-k]} \\ \tilde{c}_{[d]} =& -\sum_{k=1}^{d} \tilde{x}_{[k]} s_{[d-k]} \qquad d = 1, \ldots, D-1 \ , \end{aligned}$$

*is a recurrence for*

$$\begin{aligned} [s]_D =& E_D(\sin)([x]_D) \\ [c]_D =& E_D(\cos)([x]_D) \ , \end{aligned}$$

*Proof.* Differentiating the sine w.r.t. $t$ one obtains $\dot{s} = \frac{\partial s}{\partial x}\dot{x} = c\dot{x}$. Similarly, on finds for the cosine $\dot{c} = -s\dot{x}$. Hence one can use Lemma A.0.1. □

*Remark.* The above Lemma suggest that a specialized sincos function should be used to evaluate both the sine and cosine at once when both the sine and the cosine are required.

**Lemma A.0.6** (Tangent in UTP arithmetic)**.** *Let $[x]_D \in \mathbb{R}[T]/(T^D)$ be given, then*

$$\begin{aligned}
y_0 &= \tan(x_{[0]}) \\
z_0 &= \sec^2(x_{[0]}) \\
\tilde{y}_d &= \sum_{k=1}^{d} \tilde{x}_{[k]} z_{[d-k]} \\
\tilde{z}_d &= 2 \sum_{k=1}^{d} y_{[d-k]} \tilde{y}_{[k]} \qquad d = 1, \ldots, D-1
\end{aligned}$$

*is a recurrence for*

$$\begin{aligned}
[y]_D &= E_D(\tan)([x]_D) \\
[z]_D &= E_D(\sec^2)([x]_D) \,,
\end{aligned}$$

*The notation $\tilde{y}_{[k]} := k y_{[k]}$ is used.*

*Proof.* This follows from $\frac{\partial \tan(x)}{\partial x} = \frac{1}{\cos^2(x)} = \sec^2(x) = z$ and $\frac{\partial \sec^2(x)}{\partial x} = 2\sec^2(x)\tan(x)$. Hence $\dot{y} = \frac{\partial y}{\partial x}\dot{x} = z\dot{x}$ and $\dot{z} = \frac{\partial \sec^2(x(t))}{\partial t} = 2\tan(y)\sec^2(x)\dot{x} = 2y\dot{y}$. The recurrence for $\dot{z} = 2y\dot{y}$ is $\sum_{d=0}^{\infty} \tilde{z}_{[d+1]} = 2\sum_{d=0}^{\infty}\sum_{k=0}^{d} y_{[d-k]}\tilde{y}_{[k+1]}$ and for $\dot{y} = z\dot{x}$ it follows from Lemma A.0.1. □

**Lemma A.0.7** (Arcsine in UTP arithmetic)**.** *Let $[x]_D \in \mathbb{R}[T]/(T^D)$ be given, then*

$$\begin{aligned}
y_{[0]} &= \arcsin(x_{[0]}) \\
z_{[0]} &= \cos(y_{[0]}) = \sqrt{1 - x_{[0]}^2} \\
\tilde{y}_{[d]} &= \frac{1}{z_{[0]}} \left( \tilde{x}_{[d]} - \sum_{k=1}^{d-1} z_{[d-k]} \tilde{y}_{[k]} \right) \\
\tilde{z}_{[d]} &= -\sum_{k=1}^{d} \tilde{y}_{[k]} x_{[d-k]} \qquad d = 1, \ldots, D-1
\end{aligned}$$

*is a recurrence for*

$$\begin{aligned}
[y]_D &= E_D(\arcsin)([x]_D) \\
[z]_D &= E_D(\sqrt{1-x^2})([x]_D) \,.
\end{aligned}$$

*The notation $\tilde{y}_{[k]} := k y_{[k]}$ is used.*

*Proof.* The arcsine $y = \arcsin(x)$ is specified by the defining equation $x = \sin(y)$. From $x = \sin(y)$ follows after differentiation w.r.t. $t$ the equation $\dot{x} = \cos(y)\dot{y} = z\dot{y}$ and from $z = \cos(y) = \cos(\arcsin(x)) = \sqrt{1-x^2}$ one obtains $\dot{z} = -\sin(y)\dot{y} = -x\dot{y}$. Using Lemma A.0.1 and Lemma A.0.2 leads to the result. □

**Lemma A.0.8** (Arccosine in UTP arithmetic)**.** *Let* $[x]_D \in \mathbb{R}[T]/(T^D)$ *be given, then*

$$\begin{aligned}
y_{[0]} =& \arccos(x_{[0]}) \\
z_{[0]} =& -\sin(y_{[0]}) = -\sqrt{1 - x_{[0]}^2} \\
\tilde{y}_{[d]} =& \frac{1}{z_{[0]}} \left( \tilde{x}_{[d]} - \sum_{k=1}^{d-1} z_{[d-k]} \tilde{y}_{[k]} \right) \\
\tilde{z}_{[d]} =& -\sum_{k=1}^{d} \tilde{y}_{[k]} x_{[d-k]} \qquad d = 1, \ldots, D-1
\end{aligned}$$

*is a recurrence for*

$$\begin{aligned}
[y]_D =& E_D(\arccos)([x]_D) \\
[z]_D =& E_D(-\sqrt{1 - x^2})([x]_D) \, .
\end{aligned}$$

*The notation* $\tilde{y}_{[k]} := k y_{[k]}$ *is used.*

*Proof.* The arccosine $y = \arccos(x)$ is specified by the defining equation $x = \cos(y)$. From $x = \cos(y)$ follows after differentiation w.r.t. $t$ the equation $\dot{x} = -\sin(y)\dot{y} = z\dot{y}$ and from $z = -\sin(y(t)) = -\sin(\arccos(x)) = -\sqrt{1 - x^2}$ one obtains $\dot{z} = -\cos(y)\dot{y} = -x\dot{y}$. Using Lemma A.0.1 and Lemma A.0.2 one obtains the result. □

**Lemma A.0.9** (Arctangent in UTP arithmetic)**.** *Let* $[x]_D \in \mathbb{R}[T]/(T^D)$ *be given, then*

$$\begin{aligned}
y_{[0]} =& \arctan(x_{[0]}) \\
z_{[0]} =& \sec^2(y_{[0]}) = 1 + x_{[0]}^2 \\
\tilde{y}_{[d]} =& \frac{1}{z_{[0]}} \left( \tilde{x}_{[d]} - \sum_{k=1}^{d-1} z_{[d-k]} \tilde{y}_{[k]} \right) \\
\tilde{z}_{[d]} =& \sum_{k=1}^{d} 2 x_{[d-k]} \tilde{x}_{[k]} \qquad d = 1, \ldots, D-1
\end{aligned}$$

*is a recurrence for*

$$\begin{aligned}
[y]_D =& E_D(\arctan)([x]_D) \\
[z]_D =& E_D(1 + x^2)([x]_D) \, .
\end{aligned}$$

*The notation* $\tilde{y}_{[k]} := k y_{[k]}$ *is used.*

*Proof.* The arctangent $y = \arctan(x)$ is specified by the defining equation $x = \tan(y)$. Differentiating w.r.t. $t$ yields $\dot{x} = \sec^2(y)\dot{y} = z\dot{y}$ and $\dot{z} = 2\sec^2(y)\tan(y)\dot{y} = 2x\dot{x}$. Then the result follows from Lemma A.0.2, $\sum_{d=0}^{\infty} \tilde{z}_{[d+1]} t^d = \sum_{d=0}^{\infty} \sum_{k=0}^{d} 2 x_{[d-k]} \tilde{x}_{[k+1]}$ and $\sec(\arctan(x)) = \sqrt{1 + x^2}$. □

**Lemma A.0.10** (Hyperbolic Sine and Cosine in UTP arithmetic)**.** *Let* $[x]_D \in \mathbb{R}[T]/(T^D)$ *be*

*given, then*

$$\begin{aligned} s_{[0]} &= \sinh(x_{[0]}) \\ c_{[0]} &= \cosh(x_{[0]}) \\ \tilde{s}_{[d]} &= \sum_{k=1}^{d} \tilde{x}_{[k]} c_{[d-k]} \\ \tilde{c}_{[d+1]} &= \sum_{k=1}^{d} \tilde{x}_{[k]} s_{[d-k]} \qquad d = 1, \ldots, D-1 \end{aligned}$$

*is a recurrence for*

$$\begin{aligned} [s]_D &= E_D(\sinh)([x]_D) \\ [c]_D &= E_D(\cosh)([x]_D)\ . \end{aligned}$$

*The notation* $\tilde{y}_{[k]} := k y_{[k]}$ *is used.*

*Proof.* After differentiation of $s(t) = \sinh(x(t))$ and $c(t) = \cosh(x(t))$ w.r.t. $t$ one obtains $\dot{s} = c\dot{x}$ and $\dot{c} = s\dot{x}$. Using Lemma A.0.1 one obtains the result. □

**Lemma A.0.11** (Hyperbolic Tangent in UTP arithmetic)**.** *Let* $[x]_D \in \mathbb{R}[T]/(T^D)$ *be given, then*

$$\begin{aligned} y_{[0]} &= \tanh(x_{[0]}) \\ z_{[0]} &= 1 - \tanh^2(x_{[0]}) = \operatorname{sech}^2(x_{[0]}) \\ \tilde{y}_{[d]} &= \sum_{k=1}^{d} \tilde{x}_{[k]} z_{[d-k]} \\ \tilde{z}_{[d]} &= -\sum_{k=1}^{d} 2 y_{[d-k]} \tilde{y}_{[k]} \qquad d = 0, \ldots, D-1 \end{aligned}$$

*is a recurrence for*

$$\begin{aligned} [y]_D &= E_D(\tanh)([x]_D) \\ [z]_D &= E_D(\operatorname{sech}^2)([x]_D)\ . \end{aligned}$$

*The notation* $\tilde{y}_{[k]} := k y_{[k]}$ *is used.*

*Proof.* Differentiating $y$ w.r.t. yields $\dot{y} = (1 - \tanh^2(x))\dot{x} = z\dot{x}$ and $\dot{z} = -2\tanh(x)(1 - \tanh^2(x))\dot{x} = -2y\dot{y}$. By application of Lemma A.0.1 one finds the desired result. □

# B. Basic Matrix Calculus Identities

To derive the results in Chapter 3 one requires several matrix calculus identities that are not given in standard textbooks. They can be easily proved but may not be obvious at first sight. The purpose of this section is to collect these identities.

**Lemma B.0.12.** *Every matrix $A \in \mathbb{R}^{N\times N}$ can be written as the sum of a symmetric matrix $S = \frac{1}{2}(A + A^T)$ and an antisymmetric matrix $X = \frac{1}{2}(A - A^T)$, i.e.*

$$A = S + X$$

*Proof.* $A = \frac{1}{2}(A + A^T + A - A^T) = \frac{1}{2}(A + A^T) + \frac{1}{2}(A - A^T) = S + X.$ □

**Lemma B.0.13.** *Let $A, B \in \mathbb{R}^{N\times N}$ be lower triangular matrices and $\mathbf{1} \in \mathbb{R}^{N\times N}$ the identity matrix. Then the following expression holds:*

$$\mathbf{1} \circ (AB) = (\mathbf{1} \circ A)(\mathbf{1} \circ B) \ .$$

*Proof.* $(\mathbf{1} \circ (AB))_{ij} = \delta_{ij} \sum_{k=1}^{N} A_{ik}\delta_{i\geq k}B_{kj}\delta_{k\geq j} = \delta_{ij} \sum_{j\leq k\leq i} A_{ik}B_{kj} = \delta_{ij}A_{jj}B_{jj} = ((\mathbf{1} \circ A)(\mathbf{1} \circ B))_{ij}$ □

**Lemma B.0.14.** *The formula*

$$\mathbf{1} \circ (A^T) = \mathbf{1} \circ A$$

*holds for all matrices $A \in \mathbb{R}^{N\times N}$.*

*Proof.* $(\mathbf{1} \circ (A^T))_{ij} = \delta_{ij}A_{ji} = \delta_{ij}A_{ij} = (\mathbf{1} \circ A)_{ij}$ □

**Lemma B.0.15.** *Let $A \in \mathbb{R}^{N\times N}$ be a nonsingular lower triangular matrix. Then the formula*

$$\mathbf{1} \circ (A^{-1}) = (\mathbf{1} \circ A)^{-1}$$

*holds.*

*Proof.* Using Lemma B.0.13 one obtains $(\mathbf{1} \circ (A^{-1}))(\mathbf{1} \circ A) = \mathbf{1} \circ \mathbf{1} = \mathbf{1}$. Since the square matrices form a group, the inverse is unique. Therefore, equality between $(\mathbf{1} \circ (A^{-1})) = (\mathbf{1} \circ (A))^{-1}$ must hold. □

**Lemma B.0.16.** *Let $A \in \mathbb{R}^{N\times N}$ be strictly lower triangular and $B \in \mathbb{R}^{N\times N}$ lower triangular. Then their product $C = AB$ is strictly lower triangular.*

*Proof.* $C_{ij} = \sum_{k=1}^{N} A_{ik}\delta_{i\geq k}B_{km}\delta_{k\geq j} = \sum_{k=1}^{N} A_{ik}B_{kj}\delta_{i\geq k\geq j}$. If $j \geq i$ then there is no $k \in \mathbb{N}$ that satisfies $i \geq k \geq j$ and thus $C_{ij} = 0$ for all $j \geq i$. □

**Corollary B.0.17.** *Let $A \in \mathbb{R}^{N\times N}$ be strictly lower triangular and $D \in \mathbb{R}^{N\times N}$ diagonal. Then their product $C = AD$ is strictly lower triangular.*

**Lemma B.0.18.** *Let $A \in \mathbb{R}^{N\times N}$ and $P_L$ resp. $P_R$ as defined in Definition 3.3.3. Then*

$$(P_L \circ A)^T = P_R \circ A^T \ .$$

*Proof.* $B_{ij} := (P_L \circ A)_{ij} = A_{ij}\delta_{i>j}$ and $B^T_{ij} = B_{ji} = A_{ji}\delta_{j>i} = P_R \circ A^T$. □

**Lemma B.0.19.** *Let $X \in \mathbb{R}^{N\times N}$ be an antisymmetric matrix, i.e. $X^T = -X$ and $P_L$ defined as above. Hence, one can write*

$$X = P_L \circ X - (P_L \circ X)^T \ .$$

*Proof.* $X = P_L \circ X + P_R \circ X = P_L \circ X + (P_L \circ X^T)^T = P_L \circ X - (P_L \circ X)^T$ □

**Lemma B.0.20.** *Let $A, B, C \in \mathbb{R}^{M\times N}$. Then the identity*

$$\mathrm{tr}\left(A^T(B \circ C)\right) = \mathrm{tr}\left(C^T(B \circ A)\right)$$

*holds.*

*Proof.* $\mathrm{tr}(A^T(B \circ C)) = \sum_{i=1}^{N}\sum_{j=1}^{M} A_{ij}B_{ij}C_{ij} = \mathrm{tr}(C^T(B \circ A))$ □

**Lemma B.0.21.** *Let $A, R \in \mathbb{R}^{N\times N}$ with $R$ upper triangular. Then it holds*

$$P_L \circ ((P_L \circ A)R) = P_L \circ (AR) \ .$$

*Proof.* This follows from

$$\begin{aligned}(P_L \circ ((P_L \circ A)R))_{ij} =& \delta_{i>j}\sum_{k=0}^{N}(P_L \circ (A))_{ik}R_{kj} = \delta_{i>j}\sum_{k=0}^{N} A_{ik}\delta_{i>k}\delta_{k\leq j}R_{kj} \\ =& \delta_{i>j}\sum_{k=0}^{N} A_{ik}\delta_{k\leq j}R_{kj} = (P_L \circ (AR))_{ij} \ ,\end{aligned}$$

where it has been used that if $i > j$ and $k \leq j$ it automatically follows that $i > k$ and therefore $\delta_{i>k} = 1$ in all cases. □

**Lemma B.0.22.** *Let $b \in \mathbb{N}^{N_\mathrm{b}+1}$ define the block sizes of the block diagonal matrix $A \in \mathbb{R}^{N\times N}$. Let $\Lambda \in \mathbb{R}^{N\times N}$ be a diagonal matrix with repeated elements in the blocks defined by $b$. I.e. $\Lambda_{\mathrm{sl},\mathrm{sl}} = \lambda_{n_\mathrm{b}}\mathbf{1}$, where $\mathrm{sl} = b_{n_\mathrm{b}} : b_{n_\mathrm{b}+1} - 1$ Then $A$ and $\Lambda$ commute:*

$$0 = A\Lambda - \Lambda A \ .$$

**Lemma B.0.23.** *Let $A \in \mathbb{R}^{N\times N}$ be symmetric . Then the identity*

$$\mathrm{tr}(AB) = \mathrm{tr}(\frac{1}{2}(B + B^T)A)$$

*holds for all $B \in \mathbb{R}^{N\times N}$.*

*Proof.* $\mathrm{tr}(AB) = \sum_{ij} A_{ij}B_{ij} = \sum_{ij} \frac{1}{2}(A_{ij} + A_{ij})B_{ij} = \sum_{ij} \frac{1}{2}A_{ji}B_{ij} + \frac{1}{2}A_{ij}B_{ij} = \sum_{ij} \frac{1}{2}A_{ij}B_{ji} + \frac{1}{2}A_{ij}B_{ij} = \sum_{ij} A_{ij}\frac{1}{2}(B_{ji} + B_{ij}) = \mathrm{tr}(A\frac{1}{2}(B + B^T))$ □

**Lemma B.0.24.** *The lower, upper, strictly lower and strictly upper triangular matrices $\mathbb{R}^{N\times N}_L$, $\mathbb{R}^{N\times N}_U$, $\mathbb{R}^{N\times N}_{SL}$, $\mathbb{R}^{N\times N}_{SU}$ each form a subgroup of of $\mathbb{R}^{N\times N}$.*

**Lemma B.0.25.** *The set of strictly lower triangular matrices $\mathbb{R}^{N\times N}_{SL}$ form a two-sided ideal of the lower triangular matrices $\mathbb{R}^{N\times N}_L$. That means for $S_1, S_2 \in \mathbb{R}^{N\times N}_{SL}$ and $L \in \mathbb{R}^{N\times N}_L$ it holds*

*that*

$$\begin{aligned} S_1 + S_2 &\in \mathbb{R}_{SL}^{N\times N} \\ S_1 L &\in \mathbb{R}_{SL}^{N\times N} \\ L S_1 &\in \mathbb{R}_{SL}^{N\times N} \ . \end{aligned}$$

*Analogical for the upper triangular matrices.*

*Proof.* Set $S \equiv S_1$ for notational convenience. The first condition is clear.

$$\begin{aligned} (SL)_{nn} &= \sum_{k=1}^{N} S_{nk} L_{kn} = \sum_{k=1}^{N} \delta_{n>k} S_{nk} \delta_{k\geq n} L_{kn} = 0 \\ (LS)_{nn} &= \sum_{k=1}^{N} L_{nk} S_{kn} = \sum_{k=1}^{N} \delta_{n\geq k} L_{nk} \delta_{k>n} S_{kn} = 0 \end{aligned}$$

□

# C. Interfacing ADOL-C and Tapenade

Important standard AD tools are Tapenade (c.f. Hascoët and Pascual [2004]) and ADOL-C (c.f. Griewank et al. [1999]). The approach is quite different. While Tapenade is a source code transformation tool to generate codes for first order derivatives, ADOL-C computes in univariate Taylor polynomial arithmetic. The tools also cover different programming languages. Tapenade currently supports FORTRAN and C but cannot be used for C++ codes. ADOL-C only works for C++ codes. To differentiate programs where parts are written in FORTRAN and other parts in C++ one has to connect the derivative accumulation. In this section we show how ADOL-C can be connected to Tapenade for the special case of a combined forward/reverse accumulation for second order derivatives. More precisely, let the nominal function be $f : \mathbb{R}^N \to \mathbb{R}^M$, $y_{[0]} = f(x_{[0]})$.

**The Tangent Mode** yields a program that computes

$$\begin{aligned} y_{[0]} =& f(x_{[0]}) \\ y_{[1]} =& f'(x_{[0]})x_{[1]} \ , \end{aligned}$$

where $f'(x_{[0]}) \in \mathbb{R}^{M\times N}$, $x_{[0]}, x_{[1]} \in \mathbb{R}^N$ and $y_{[0]}, y_{[1]} \in \mathbb{R}^M$. This can be written as the function $g : \mathbb{R}^N \times \mathbb{R}^N \to \mathbb{R}^M \times \mathbb{R}^M$

$$g(x_{[0]}, x_{[1]}) = (y_{[0]}, y_{[1]}) = (f(x_{[0]}), f'(x_{[0]})x_{[1]}) \ .$$

**The Reverse mode** yields a program that computes

$$\bar{x}_{[0]}^T = \bar{y}_{[0]}^T (f'(x_{[0]})) \ .$$

The function can be written as a function $h : \mathbb{R}^N \times \mathbb{R}^M \to \times\mathbb{R}^N$

$$h(x_{[0]}, \bar{y}_{[0]}) = \bar{x}_{[0]}^T = \bar{y}_{[0]}^T \frac{\partial f}{\partial x}(x_{[0]}) \ .$$

This is motivated from the pullback

$$\bar{y}_{[0]}^T \dot{y}_{[0]} = \bar{y}_{[0]}^T \frac{\partial f}{\partial x}(x_{[0]})\dot{x}_{[0]} = \bar{x}_0^T \dot{x}_{[0]} \ .$$

Sometimes the reverse mode is regarded as a function $\tilde{h} : \mathbb{R}^N \times \mathbb{R}^M \to \times\mathbb{R}^M \times \mathbb{R}^N$ that also computes $y_{[0]} = f(x_{[0]})$ in the process. Since it is not necessary in the following we use the function $h$ to keep the notation easy to read.

**Reverse on Tangent** is the combination of first tangent then reverse. I.e., application of the reverse mode transformation to $g(x_{[0]}, x_{[1]})$:

$$\begin{aligned}\langle \tilde{g} | \dot{g} \rangle =& \tilde{y}_{[0]}^T \dot{y}_0 + \tilde{y}_{[1]}^T \dot{y}_{[1]} \\ =& \tilde{y}_{[0]}^T f'(x_{[0]}) \dot{x}_{[0]} + \tilde{y}_{[1]}^T f''(x_{[0]}) x_{[1]} \dot{x}_{[0]} + \tilde{y}_{[1]}^T f'(x_{[0]}) \dot{x}_{[1]} \\ =& (\tilde{y}_{[0]}^T f'(x_{[0]}) + \tilde{y}_{[1]}^T f''(x_{[0]}) x_{[1]}) \dot{x}_{[0]} + (\tilde{y}_{[1]}^T f'(x_{[0]})) \dot{x}_{[1]} \\ =& \tilde{x}_{[0]}^T \dot{x}_{[0]} + \tilde{x}_{[1]}^T \dot{x}_{[1]} \ ,\end{aligned}$$

where $\langle a | b \rangle$ is used as linear form on a product space. I.e.,

$$\begin{aligned}\tilde{x}_{[0]} =& \tilde{y}_{[0]} f'(x_{[0]}) + \tilde{y}_{[1]} f''(x_{[0]}) x_{[1]} \\ \tilde{x}_{[1]} =& \tilde{y}_{[1]} f'(x_{[0]}) \ .\end{aligned}$$

On the other hand, ADOL-C computes

$$\begin{aligned}[\bar{y}_{[0]}^T, \bar{y}_{[1]}^T][\dot{y}_{[0]}, \dot{y}_{[1]}] \overset{2}{=}& [\bar{y}_{[0]}^T, \bar{y}_{[1]}^T] E_2(f')([x_{[0]} + x_{[1]}])[\dot{x}]_2 \\ \overset{2}{=}& [\bar{y}_{[0]}^T, \bar{y}_{[1]}^T][f'(x_{[0]}), f''(x_{[0]}) x_{[1]}][\dot{x}] \\ \overset{2}{=}& [\bar{y}_{[0]}^T f'(x_{[0]}), \bar{y}_{[0]}^T f'' x_{[1]} + \bar{y}_{[1]}^T f'(x_{[0]})][\dot{x}] \ ,\end{aligned}$$

i.e.

$$\begin{aligned}\bar{x}_{[0]}^T =& \bar{y}_{[0]}^T f'(x_{[0]}) \\ \bar{x}_{[1]}^T =& \bar{y}_{[1]}^T f'(x_{[0]}) + \bar{y}_{[0]}^T f''(x_{[0]}) x_{[1]} \ .\end{aligned}$$

That means that one can compute parts of the combined forward reverse mode by using a combination of Tapenade and ADOL-C by sorting the coefficients in reverse direction. This is nothing else as a special case of Proposition 2.5.4. However, in practice one uses a vectorized version of the tangent mode where $P \in \mathbb{N}$ directional derivatives are evaluated at once. Tapenade uses a $x_{[1]} \in \mathbb{R}^{N \times P}$ matrix for that. The catch is that once a new function has been generated, Tapenade considers the new function $g(x_{[0]}, x_{[1]})$ as a matrix valued function. In particular, one obtains an output $y_{[1]} \in \mathbb{R}^{M \times P}$. E.g., look at $\langle \bar{y}_{[1]} | \dot{y}_{[1]} \rangle = \sum_{m=1}^{M} \sum_{p=1}^{P} \bar{y}_{mp;[1]} \dot{y}_{mp;[1]} = \sum_{nl} \sum_{mp} \bar{y}_{mp;[1]} \frac{\partial y_{mp;[1]}}{\partial x_{nl;[1]}} \dot{x}_{nl;[1]} + \sum_n \sum_{mp} \bar{y}_{mp;[1]} \frac{\partial y_{mp;[1]}}{\partial x_{n;[0]}} \dot{x}_{n;[0]} = \sum_{nl} \bar{x}_{nl;[1]} \dot{x}_{nl;[1]} + \sum_n \bar{x}_{n;[0]} \dot{x}_{n;[0]}$. To put it into words: $\bar{x}_{[0]}$ contains a linear combination of several directional derivatives. This is not what is desired.

**Tangent on Reverse** is first the application of the reverse mode. The newly generated function is then further differentiated using the tangent mode. I.e., one obtains a program that computes

$$\begin{aligned}\bar{x}_{[0]} =& h(x_{[0]}, \bar{y}_0) = (f'(x_{[0]}))^T \bar{y}_{[0]} \\ \bar{x}_{[1]} =& h'(x_{[0]}, \bar{y}_{[0]}) \begin{pmatrix} x_{[1]}^T \\ \bar{y}_{[1]} \end{pmatrix} = \bar{y}_{[0]}^T f''(x_{[0]}) x_{[1]} + \bar{y}_{[1]}^T f'(x_{[0]}) \ .\end{aligned}$$

This coincides with what ADOL-C computes.

# D. Illustrative Examples

Sometimes it is instructive to consider simple but sufficiently complex examples to verify numerical algorithms. Here, several of such problems are briefly discussed.

## D.1. Shooting Problem

The purpose of this section is to serve as a simplistic example of two point boundary value problem. It is described how problems with unknown end time $t_e$ can be reformulated by introduction of an additional parameter to the model function. Also, it is shown how the boundary value problem is reformulated as a single shooting problem: i.e., a nonlinear program with initial value problem constraint.

It is the goal to hit a point TU with a cannon located at the origin $(0, 0)$. The initial velocity is fixed at $v_0$ and is only possible to vary the angle $\theta$ at which the projectile is ejected. The movement of the projectile is defined by a second order, nonlinear ordinary differential equation

$$\begin{cases} \ddot{x} = \begin{pmatrix} 0 \\ -a_G \end{pmatrix} - \frac{1}{2m} c_w A \rho \|\dot{x}\| \dot{x} \\ x(0) = x_0 \ , x(t_e) = \text{TU} \ , \end{cases}$$

where $x_0 = x_0(\theta, p) \in \mathbb{R}^2$. Guesses of the parameters $p = (a_g, v_0, c_w, \rho, m, A) \in \mathbb{R}^6$ are collected in Table D.1. $x(t) \in \mathbb{R}^2$ and $v(t) := \dot{x}(t)$ denote the position resp. the velocity of the projectile. Written as first order ODE

$$\dot{y} = \begin{pmatrix} \dot{x} \\ \dot{v} \end{pmatrix} = \begin{pmatrix} v \\ \begin{pmatrix} 0 \\ -a_g \end{pmatrix} - \frac{1}{2m} A \rho c_w \|v\| v \end{pmatrix} = f(t, y) \in \mathbb{R}^4 \ . \tag{D.1}$$

- The first step is to remove the dependency on the unknown end time $t_e$. Transformation of the time $\tau = \frac{t - t_0}{t_e - t_0} \in [0, 1]$, $\frac{\partial}{\partial t} = \frac{\partial \tau}{\partial t} \frac{\partial}{\partial \tau} = \frac{1}{t_e - t_0} \frac{\partial}{\partial \tau}$ yields the ODE

$$\begin{aligned} \frac{\mathrm{d}y}{\mathrm{d}\tau} &= (t_e - t_0) f((t_e - t_0)\tau + t_0, y) \\ &= \tilde{f}(\tau, y, t_e) \\ y(0) &= y_0 \ , \end{aligned}$$

  and the integration time horizon is fixed to $[0, 1]$. One can see that $t_e$ is now a parameter in the model function $\tilde{f}$.

- Now it is possible to compute the solution $y(t; y_0, t_e, p)$ given the initial conditions

$$y_0 = y_0(\theta) = \begin{pmatrix} \cos(\theta) v_0 \\ \sin(\theta) v_0 \\ 0 \\ 0 \end{pmatrix} \ .$$

- To satisfy the boundary value conditions one has to solve the nonlinear equation

$$0 = F(\theta, t_e) = x(1; y_0(\theta), t_e) - \mathrm{TU} \ ,$$

  for instance with Newton's method. One can see that there are two equations and two unknowns. I.e., one computes a new iterate by

$$(\theta, t_e)^+ = (\theta, t_e) - (J(\theta, t_e))^{-1} F(\theta, t_e)$$

- The Jacobian $J(\theta, t_e) = \frac{\partial F}{\partial(\theta, t_e)}(\theta, t_e)$ can be computed in univariate Taylor polynomial arithmetic

$$F_{[0]} + F_{[1]}T = E_D(F)(\theta_{[0]} + \{1, 0\}T, t_{[0],e} + \{0, 1\}T),$$

  i.e., $J = F_{[1]}$.

- That means one has to compute all operations of the time integration in univariate Taylor polynomial arithmetic. In Appendix D.4 it is shown how this can be done for the implicit Euler.

- In Figure D.1 the solution trajectories of all Newton iterates are shown. One can see that full-step Newton's method converges in just a few steps.

| constant | notation | value | unit |
|---|---|---|---|
| Earth's gravity constant | $a_g$ | 9.81 | $\frac{m}{s^2}$ |
| initial velocity | $v_0$ | 1600 | $\frac{m}{s}$ |
| air drag factor | $c_w$ | 0.15 | |
| density of air | $\rho$ | 1.3 | $\frac{kg}{m^3}$ |
| mass of projectile | $m$ | 194 | $kg$ |
| area of projectile | $A$ | $\pi(0.105)^2$ | $m^2$ |

Table D.1.: The parameters which have been used in the simulation. The true values have been lost during war and only estimates are available. The values have been adapted from Wikipedia.

## D.2. High-Dimensional Integration by Using Method of Moments

Consider the problem of computing the integral

$$I(f) = \int_{U \in \mathbb{R}^N} f(x) \mathrm{d}x \ ,$$

where $f : \mathbb{R}^N \to \mathbb{R}$ is an entire, i.e., everywhere analytic, function and $U$ a compact region about $x_0 \in \mathbb{R}^N$. The need of evaluation of such high-dimensional integrals often appears in statistics. E.g. to compute the expectation value

$$\mathbb{E}[f(x)] = \int_{\mathbb{R}^N} f(x) p(x) \mathrm{d}x$$

of a nonlinear function $f(x)$, where $x$ are outcomes of a random vector with probability density function $p(x)$ of compact support. One can compute the Taylor series of $f$ about a point $x_0$

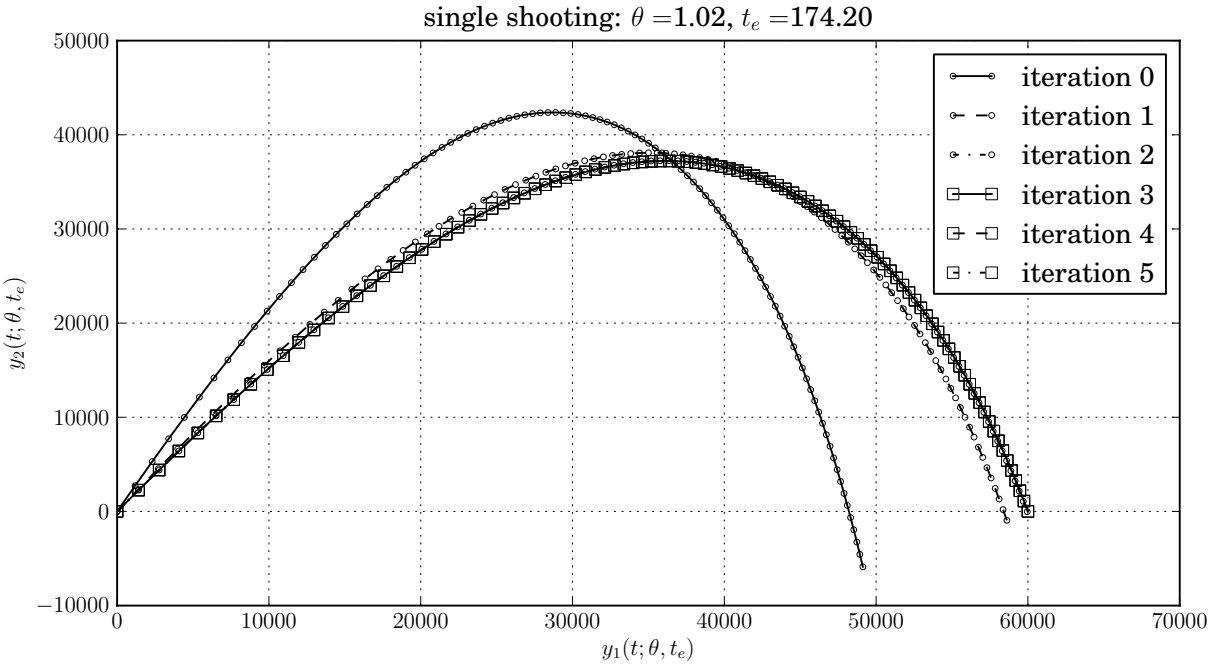



Figure D.1.: This figure shows solution trajectories suggested by Newton's method of the shooting problem (Chapter D.1). One can see that Newton's method converges very quickly and after just 5 iterations the residual norm is about $10^{-12}$. The target unit TU is here at $(y_1, y_2) = (60000, 0)$.

and obtains

$$\mathbb{E}[f(x)] = \int_{\mathbb{R}^N} \sum_{d=0}^{\infty} \sum_{|\mathbf{i}|=d} y_{\mathbf{i}} \cdot (x - x_0)^{\mathbf{i}} p(x) \mathrm{d}x = \sum_{d=0}^{\infty} \sum_{|\mathbf{i}|=d} y_{\mathbf{i}} \int_{\mathbb{R}^N} (x - x_0)^{\mathbf{i}} p(x) \mathrm{d}x \ .$$

The integrals $m_{\mathbf{i}} := \int_{\mathbb{R}^N} (x - x_0)^{\mathbf{i}} p(x) \mathrm{d}x$ are called *central moments* and typically are analytically known for the important probability distributions as e.g. the normal distribution. The idea is that one only computes the sum up to a certain degree $d$ and obtains an approximation of the expectation value. E.g., if the function was in fact quadratic or almost so, one could expect that a second order model would suffice to compute the derivative accurately. Since for the full polynomial approximation of degree $d$ requires the computation $\binom{N+d}{d} = \alpha(d, N) N^d$ coefficients. For fixed $d$ one has $\lim_{N \to \infty} \alpha(d, N) = 1$. Hence one has the nice property that the integration is only polynomial in $N$ and not exponential. I.e., using the method of moments one could avoid the curse of dimensionality. On the other hand, the problem is exponential in $d$.

For general composite functions, there are several reasons that make the approach difficult to handle. The first is the unknown degree $d$ that has to be put into relation with the integration error. Then, in practice one may be confronted with composite functions that are not entire, e.g. $f(x) = \frac{1}{1+x^2}$ which has two poles at $x = \pm i$ in the complex plane. Therefore, the radius of convergence $\rho$ may be smaller than the support of $p(x)$ and the method breaks down. One can of course try to use a partition of unity to transform the problem such that the support is smaller than the radius of convergence. In the one dimensional case this decomposition has been investigated by Menshikova [2010]. See [Amann and Escher, 1999, Section IV.4, example 4.15 ] for a short discussion and references.

## D.3. Ray Tracing

Consider a cylindric mirror in two dimensions where a laser beam enters at $x^{(0)} = (0, -1)^T$ with initial direction $v^{(0)}$. The mirror is described by the unit circle $0 = g(x) = \|x\|^2 - 1 = x_1^2 + x_2^2 - 1$,where $x \in \mathbb{R}^2$. One is interested in the points where the laser beam is reflected. The time evolution is described by $x(t) = x^{(k)} + v^{(k)}t$. To find the next reflection point $x^+ \equiv x^{(k+1)}$ given $x \equiv x^{(k)}$ one has to find $t$ to satisfy $0 = g(x(t))$. After a short calculation one obtains $x^+ = x + \left(\sqrt{\left(\frac{x^T v}{\|v\|^2}\right)^2 - \frac{\|x\|^2-1}{\|v\|^2}} - \frac{x^T v}{\|v\|^2}\right) v$. The reflection changes the direction of $v$ and is described by the reflector $P(x^+) = \mathbf{I} - 2\frac{ww^T}{\|w\|^2}$, where $w = w(x^+) = \nabla_x g(x^+)$ points away from the center. This is geometrically motivated and satisfies $w^T P v = -w^T v$. That means that the overall progression of the reflection points is given by

$$\begin{pmatrix} x^+ \\ v^+ \end{pmatrix} = F(x, v) = \begin{pmatrix} x + \left(\sqrt{\left(\frac{x^T v}{\|v\|^2}\right)^2 - \frac{\|x\|^2-1}{\|v\|^2}} - \frac{x^T v}{\|v\|^2}\right) v \\ P(x^+)v \end{pmatrix} .$$

One would like to know the sensitivity of the end point w.r.t. the initial direction $v^{(0)}$. This can be done easily in the forward mode by univariate Taylor polynomial arithmetic. Explicitly, one generalizes to $[x]_2 \in \mathbb{R}^2[T]/(T^2)$ and $[v]_2 \in \mathbb{R}^2[T]/(T^2)$. The corresponding code is shown in Listing D.1 and the output is plotted in Figure D.2.

```
Nr = 20
# initial laser beam
x0 = array([0,-1.]); v0 = array([0.1,1])
def F(x,v):
    """ computes the next reflection point and direction"""
    c = dot(v,v)
    x2 = x + v*(sqrt( (dot(x,v)/c)**2 - (dot(x,x) - 1.)/c) - dot(x,v)/c)
    w = x2
    v2 = (v - 2* w * dot(w, v)/dot(w,w))
    return x2,v2

# AD solution
# ~~~~~~~~~~~
import taylorpoly; from taylorpoly import UTPS
x = numpy.array([UTPS([x0[0],0]),UTPS([x0[1],0])])
v = numpy.array([UTPS([v0[0],1]),UTPS([v0[1],0])])

for nr in range(Nr):
    x,v = F(x,v)
    AD_dxdv = array([x[0].data[1], x[1].data[1]])
    print 'AD dx/dv = ', AD_dxdv
```

Listing D.1: Raytracing of a laser beam in a circular mirror. The number `Nr` is the number of reflections.

As output one obtains at each reflection the derivative. One can see that the the derivatives are propagated in forward mode.

```
AD dx/dv =  [ 1.94098618 -0.39211842]
AD dx/dv =  [-3.64980719  1.53741479]
AD dx/dv =  [ 4.90964281 -3.34455755]
etc.
```

Listing D.2: Output of Listing D.1.

## D.4. Differentiation of the Implicit and Explicit Euler

The purpose of this section is to demonstrate how numerical integration schemes can be evaluated in univariate Taylor polynomial arithmetic. The idea is directly related to the work of

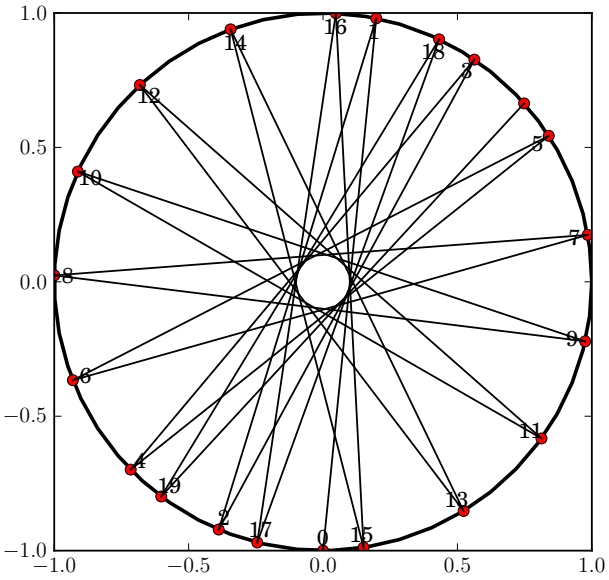



Figure D.2.: This figure shows how a laser beam propagates in a cylindric mirror. The numbers show the sequence of points where the beam is reflected.

Albersmeyer who applies the techniques to a BDF integration scheme. See Albersmeyer and Bock [2008] for an early work. It is the goal is to compute derivatives of the form $\frac{\partial x(t)}{\partial p}$, where $x(t) \equiv x(t; x_0, p) \in \mathbb{R}^{N_x}$ is solution of the ordinary differential equation

$$\begin{aligned} \dot{x}(t) &= f(t, x, p) \\ x(0) &= x_0(p) \ , \end{aligned}$$

where the initial values $x(0) \equiv x_0(p)$ is a function depending on some parameter $p \in \mathbb{R}^{N_p}$. To give an explicit example, consider the harmonic oscillator described by the differential equation

$$\dot{x}(t) = \begin{pmatrix} x_2 \\ -px_1 \end{pmatrix} \ .$$

To obtain the derivatives $\frac{\partial x(t)}{\partial p}$ one could in principle differentiate the differential equation w.r.t. $p$ to obtain the *variational differential equation*

$$\begin{aligned} \dot{x}(t) =& f(t, x, p) \\ \dot{x}_p(t) =& \frac{\partial f}{\partial x}(t, x, p)\frac{\partial x}{\partial p} + \frac{\partial f}{\partial p}(t, x, p) \\ x(0) =& x_0(p) \\ x_p(0) =& \frac{\partial x_0}{\partial p}(p) \ , \end{aligned}$$

and provide these equations to a standard solver. However, a lot of structure is neglected, doubling the problem size $N_x$. Doubling the problem size generally results in a much higher runtime. Consider for instance an implicit integration scheme that needs to solve a linear system of equations at each step, which is a $\mathcal{O}(N_x^3)$ operation for dense matrices.

**Explicit Euler** In this integration method one discretizes the time derivative with finite differences and obtains

$$x_{k+1} = (t_{k+1} - t_k) f(t_k, x_k, p) + x_k$$

To obtain derivatives, the algorithm can be evaluated in UTP arithmetic (c.f. Section 2.3), i.e., compute

$$[x_{k+1}]_D = (t_{k+1} - t_k) E_D(f)(t_k, [x_k]_D, [p]_D) + [x_k]_D \ ,$$

where $[x]_D = [x_0, 0, \dots]$ and $[p]_D = [p, 1, 0, \dots]$. To check that this method indeed generates the correct solution, the output of this integration scheme is compared to the analytical solution. The result is depicted in Figure D.3.

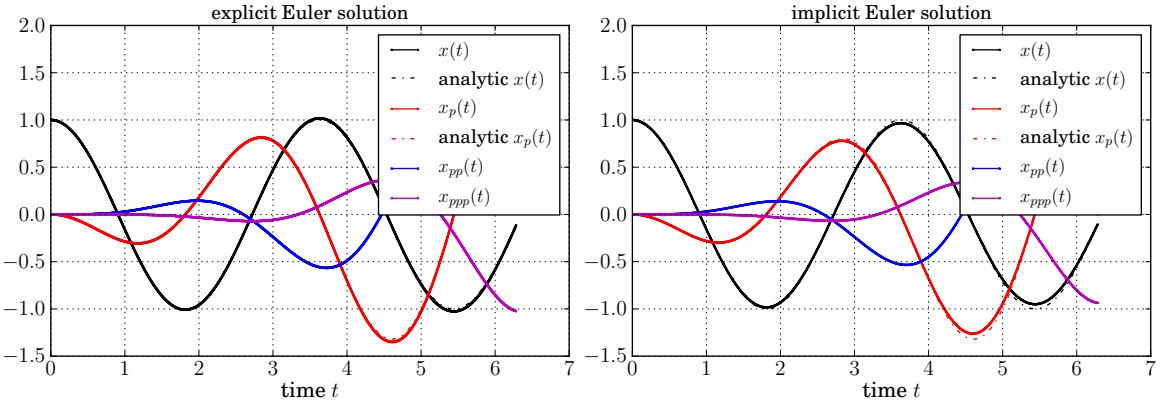



Figure D.3.: One can see that the explict as well as the implicit Euler evaluated in UTP arithmetic generated a solution $x(t)$, $x_p(t) = \frac{\partial x}{\partial p}(t)$ and $x_{pp}(t) = \frac{\partial^2 x}{\partial p^2}(t)$ that are in good agreement with the analytically derived solution. The stepsize was constant and of order $10^{-2}$.

**Implicit Euler** In practice, ODEs are often stiff and implicit integration schemes are used. To illustrate the challenges of implicit schemes it is shown how the implicit Euler method can be evaluated in UTP arithmetic. The ODE is discretized in time as

$$\begin{aligned} x_{k+1} - x_k =& (t_{k+1} - t_k) f(t_{k+1}, x_{k+1}, p) \\ \Leftrightarrow 0 =& F(x_{k+1}, x_k, t_{k+1}, t_k, p) := (t_{k+1} - t_k) f(t_{k+1}, x_{k+1}, p) - x_{k+1} + x_k \ , \end{aligned}$$

i.e., one has to solve a nonlinear system at each step. The task at hand is to solve this implicit function in UTP arithmetic, i.e. given a nonlinear system described by $0 = F(x, y)$, where $x \in \mathbb{R}_x^N$ is input and $y \in \mathbb{R}_y^N$ is output, solve

$$0 \overset{D}{=} E_D(F)([x]_D, [y]_D) \ .$$

There are several possibilities to solve the above equation. For instance one can first solve the nominal problem $0 = F(x_0, y_0)$ and then successively compute the higher-order coefficients $y_d, d = 1, \dots, D-1$ using Newton-Hensel lifting. To find the solution of the nominal problem one can use Newton's method. I.e., given an initial guess for $y_0$ compute an update $\delta y$. In other words, iterate

$$\begin{aligned} \delta y =& -(F_y(x, y))^{-1} F(x, y) \\ y =& y + \delta y \end{aligned}$$

Once $y$ is known one can find the higher-order coefficients by using Newton-Hensel lifting

$$y_d T^d = -(F_y(x, y))^{-1} F([y]_d, [x]_{d+1}) \mod T^{d+1} \ . \tag{D.2}$$

It would also be possible to solve the system $0 = F(x, y)$ with an inexact Jacobian and applied a piggy-back approach to compute the higher-order coefficients as explained in [Griewank and Walther, 2008, Section 15.2].

The complete procedure is shown in Listing D.3.

```
import numpy; from numpy import sin,cos; from algopy import UTPM, zeros

def implicit_euler(f_fcn, x0, ts, p):
    """ implicit euler with fixed stepsizes, using Newton's method to solve
    the occuring implicit system of nonlinear equations
    """

    def F_fcn(x_new, x, t_new, t, p):
        """ implicit function to solve:  0 = F(x_new, x, t_new, t_old)"""
        return (t_new - t) * f_fcn(t_new, x_new, p) - x_new + x

    def J_fcn(x_new, x, t_new, t, p):
        """ computes the Jacobian of F_fcn
        all inputs are double arrays
        """
        y = UTPM(numpy.zeros((D,N,N)))
        y.data[0,:]   = x_new
        y.data[1,:,:] = numpy.eye(N)
        F = F_fcn(y, x, t_new, t, p)
        return F.data[1,:,:].T


    x = x0.copy()
    D,P,N = x.data.shape
    x_new = x.copy()

    x_list = [x.data.copy() ]
    for nts in range(ts.size-1):
        h = ts[nts+1] - ts[nts]
        x_new.data[0,...] = x.data[0,...]
        x_new.data[1:,...] = 0

        # compute the Jacobian at x
        J = J_fcn(x_new.data[0,0], x.data[0,0], ts[nts+1], ts[nts], p.data[0,0])

        # d=0: apply Newton's method to solve 0 = F_fcn(x_new, x, t_new, t)
        step = numpy.inf
        while step > 10**-10:
            delta_x = numpy.linalg.solve(J, F_fcn(x_new.data[0,0], x.data[0,0],
                                                  ts[nts+1], ts[nts], p.data[0,0]))
            x_new.data[0,:] -= delta_x
            step = numpy.linalg.norm(delta_x)

        # d>0: compute higher order coefficients
        J = J_fcn(x_new.data[0,0], x.data[0,0], ts[nts+1], ts[nts], p.data[0,0])
        for d in range(1,D):
            F = F_fcn(x_new, x, ts[nts+1], ts[nts], p)
            for np in range(P):
                x_new.data[d,np] = -numpy.linalg.solve(J, F.data[d,np])
        x.data[...] = x_new.data[...]
        x_list.append(x.data.copy())

    return numpy.array(x_list)
```

Listing D.3: Implicit Euler in UTP arithmetic.

# Bibliography


Jan Albersmeyer. Effiziente Ableitungserzeugung in einem adaptiven BDF-Verfahren. Master's thesis, Ruprecht-Karls-Universität Heidelberg, 2005.

Jan Albersmeyer and Hans Georg Bock. Sensitivity generation in an adaptive BDF-method. In Hans Georg Bock, Ekaterina Kostina, Hoang Xuan Phu, and Rolf Rannacher, editors, *Modeling, Simulation and Optimization of Complex Processes*, pages 15–24. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-79409-7. URL `http://dx.doi.org/10.1007/978-3-540-79409-7_2`. 10.1007/978-3-540-79409-7_2.

Jan Albersmeyer and C. Kirches. The SolvIND Webpage, 2007–. URL `http://www.solvind.org`.

Ben Altman. Higher-order automatic differentiation of multivariate functions in MATLAB, 2010. Bachelor's honors thesis, Davidson College.

Herbert Amann and Joachim Escher. *Analysis II, Second Edition*. Birkhäuser, 1999.

E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999. ISBN 0-89871-447-8 (paperback).

Joel Andersson, Boris Houska, and Moritz Diehl. Towards a Computer Algebra System with Automatic Differentiation for use with Object-Oriented modelling languages. 3rd International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools, 2010.

Alan L. Andrew and Roger C. E. Tan. Computation of derivatives of repeated eigenvalues and the corresponding eigenvectors of symmetric matrix pencils. *SIAM Journal on Matrix Analysis and Applications*, 20(1):78–100, 1998. doi: 10.1137/S0895479896304332. URL `http://link.aip.org/link/?SML/20/78/1`.

C. Annamalai. Automatic Differentiation: Theory and Implementation. `http://code.google.com/p/google-summer-of-code-2010-r-project/downloads/detail?name=Chidambaram_Annamalai.tar.gz&can=2&q=`, 2010.

Michael C. Bartholomew-Biggs, Steve Brown, Bruce Christianson, and Laurence C. W. Dixon. Automatic differentiation of algorithms. *Journal of Computational and Applied Mathematics*, 124:171–190, 2000. doi: 10.1016/S0377-0427(00)00422-2.

B. Bell. CppAD: a package for C++ algorithmic differentiation, 2010. URL `http://www.coin-or.org/CppAD`.

Daniel J. Bernstein. Multidigit multiplication for mathematicians, 2001. URL `http://cr.yp.to/papers.html#m3`.

Daniel J. Bernstein. Fast multiplication and its applications. *Algorithmic Number Theory, Lattices, Number Fields, Curves and Cryptography*, 2008.

Martin Berz. Calculus and Numerics on Levi-Civita Fields. In Martin Berz, Christian Bischof, George Corliss, and Andreas Griewank, editors, *Computational Differentiation: Techniques, Applications, and Tools*, pages 19–35, Philadelphia, PA, 1996. SIAM.

C. Bischof, G. Corliss, and A. Griewank. Structured second-and higher-order derivatives through univariate taylor series. *Optimization Methods and Software*, 2(3):211–232, 1993.

Christian H. Bischof and Mohammad R. Haghighat. Hierarchical approaches to automatic differentiation. In Martin Berz, Christian Bischof, George Corliss, and Andreas Griewank, editors, *Computational Differentiation: Techniques, Applications, and Tools*, pages 83–94. SIAM, Philadelphia, PA, 1996.

Å. Björck. *Numerical Methods for Least Squares Problems*. SIAM, Philadelphia, 1996.

H. G. Bock, S. Körkel, E. Kostina, and J. P. Schlöder. *Robustness Aspects in Parameter Estimation, Optimal Design of Experiments and Optimal Control*. Springer Berlin Heidelberg, 2007a.

Hans Georg Bock. *Randwertproblemmethoden zur Parameteridentifizierung in Systemen nichtlinearer Differentialgleichungen*. PhD thesis, Bonn, 1987.

Hans Georg Bock, E. Eich, and J P. Schlöder. Numerical solution of constrained least squares boundary value problems in differential-algebraic equations. In *In K. Strehmel (ed.): Numerical Treatment of Differential Equations. Teubner*, 1988.

Hans Georg Bock, Ekaterina Kostina, Stefan Körkel, and J. P. Schlöder. Numerical Methods for Optimal Control Problems in Design of Robust Optimal Experiments for Nonlinear Dynamic Processes. *Optimization Methods and Software (OMS) Journal*, 19(3-4):327–338, 2004.

Hans Georg Bock, Ekaterina Kostina, and Olga Kostyukova. Covariance matrices for parameter estimates of constrained parameter estimation problems. *SIAM J. Matrix Anal. Appl.*, 29(2): 626–642, 2007b. ISSN 0895-4798. doi: http://dx.doi.org/10.1137/040617893.

J. Frédéric Bonnans, J. Charles Gilbert, Claude Lemaréchal, and Claudia A. Sagastizábal. *Numerical Optimization*. Springer, Berlin, Heidelberg, New York, 1997. ISBN 3-540-35445-X.

K.E. Brenan, S.L. Campbell, and L.R. Petzold. *Numerical solution of initial-value problems in differential-algebraic equations. Unabridged, corr. republ. Unabridged, corr. republ.* Classics in Applied Mathematics. 14. Philadelphia, PA: SIAM, Society for Industrial and Applied Mathematics. x, 256 p. $ 29.50 , 1996.

H. Martin Bücker. Hierarchical algorithms for automatic differentiation, 2002.

J. Bulirsch. Die Mehrzielmethode zur numerischen Lösung von nichtlinearen Randwertproblemen und Aufgaben der optimalen Steuerung. Beitrag zum Programm Flugbahnoptimierung der Carl-Cranz-Gesellschaft, Heidelberg, 1971.

Bruce Christianson. Reverse accumulation and accurate rounding error estimates for Taylor series coefficients. *Optimization Methods and Software*, 1(1):81–94, 1991. Also appeared as Tech. Report No. NOC TR239, The Numerical Optimisation Centre, University of Hertfordshire, U.K., July 1991.

Bruce Christianson. Reverse accumulation of functions containing gradients. 1993. URL `http://hdl.handle.net/2299/4337`.

Bruce Christianson. Reverse accumulation and implicit functions. *Optimization Methods and Software*, 9(4):307–322, 1998.

Biswa Nath Datta. *Numerical Linear Algebra and Applications*. SIAM, 2nd edition, 1998.

Peter Deuflhard and F. Bornemann. Numerik von Anfangswertmethoden für gewöhnliche Differentialgleichungen. Technical report, Konrad-Zuse-Zentrum fï Informationstechnik Berlin, 1987. TR 89-2, März 1989.

Peter Deuflhard and Folkmar Bornemann. *Scientific computing with ordinary differential equations*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002. ISBN 0-387-95462-7.

Richard Dudley. Mathematical Statistics (lecture script), 2003. URL `http://ocw.mit.edu/courses/mathematics/18-466-mathematical-statistics-spring-2003/lecture-notes`.

Stephen Duffull, Nick Denman, John Eccleston, and Hui Kimko. WinPOPT/POPT homepage, 2009–2010. URL `http://www.winpopt.com/index.htm`.

Christodoulos A. Floudas. *Nonlinear and mixed-integer optimization. Fundamentals and applications.* Oxford: Oxford Univ. Press. xv, 462 p., 1995.

Gaia Franceschini and Sandro Macchietto. Model-based design of experiments for parameter precision: State of the art. *Chemical Engineering Science*, 63(19):4846–4872, 2008. ISSN 0009-2509. doi: DOI:10.1016/j.ces.2007.11.034. URL `http://www.sciencedirect.com/science/article/B6TFK-4R8MDSW-5/2/8de613210a7ae4e03e8e9580e8cce821`. Model-Based Experimental Analysis.

Mariano Gasca and Thomas Sauer. Polynomial interpolation in several variables. *Adv. Comput. Math*, 12:377–410, 2000.

David M. Gay. Automatic differentiation of nonlinear AMPL models. In Andreas Griewank and George F. Corliss, editors, *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*, pages 61–73. SIAM, Philadelphia, PA, 1991. ISBN 0–89871–284–X.

Mike B. Giles. An extended collection of matrix derivative results for forward and reverse mode automatic differentiation. Technical report, Oxford University Computing Laboratory, 2007. URL `ftp://ftp.comlab.ox.ac.uk/pub/Documents/techreports/NA-08-01.pdf`. Report no 08/01.

Mike B. Giles. Collected matrix derivative results for forward and reverse mode algorithmic differentiation. In Christian H. Bischof, H. Martin Bücker, Paul D. Hovland, Uwe Naumann, and J. Utke, editors, *Advances in Automatic Differentiation*, pages 35–44. Springer, 2008. ISBN 978-3-540-68935-5.

G. H. Golub and V. Pereyra. The differentiation of pseudo-inverses and nonlinear least squares problems whose variables separate. *SIAM Journal on Numerical Analysis*, 10(2):413–432, 1973. doi: 10.1137/0710036. URL `http://link.aip.org/link/?SNA/10/413/1`.

Andreas Griewank. Achieving Logarithmic Growth of Temporal and Spatial Complexity in Reverse Automatic Differentiation. *Optimization Methods and Software*, 1:35–54, 1992.

Andreas Griewank. A mathematical view of automatic differentiation. In *Acta Numerica*, volume 12, pages 321–398. Cambridge University Press, 2003. doi: 10.1017/S0962492902000132.

Andreas Griewank and Andrea Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Number 105 in Other Titles in Applied Mathematics. SIAM, Philadelphia, PA, 2nd edition, 2008. ISBN 978–0–898716–59–7.

Andreas Griewank, David Juedes, H. Mitev, Jean Utke, Olaf Vogel, and Andrea Walther. ADOL-C: A package for the automatic differentiation of algorithms written in C/C++. Technical report, Institute of Scientific Computing, Technical University Dresden, 1999. Updated version of the paper published in *ACM Trans. Math. Software* 22, 1996, 131–167.

Andreas Griewank, Jean Utke, and Andrea Walther. Evaluating higher derivative tensors by forward propagation of univariate Taylor series. *Mathematics of Computation*, 69:1117–1130, 2000.

Andreas Griewank, Lutz Lehmann, Hernan Leovey, and Marat Zilberman. Automatic evaluation of cross-derivatives. *AMS*, 2009.

Brian Guenter. Efficient symbolic differentiation for graphics applications. *ACM Trans. Graph.*, 26(3):108, 2007. ISSN 0730-0301. doi: http://doi.acm.org/10.1145/1276377.1276512.

Luca Guerrieri. Automatic differentiation in practice: An application to the Estimation of SDGE models. `http://ice.uchicago.edu/2009_presentations/Guerrieri_slides.pdf`.

Ernst Hairer, C. Lubich, and Gerhard Wanner. *Geometric Numerical Integration: Structure-Preserving Algorithms for Ordinary Differential Equations*. 2002.

L. Hascoët and V. Pascual. TAPENADE 2.1 user's guide, 2004. URL `http://www.inria.fr/rrrt/rt-0300.html`.

M.J.R. Healy. *Matrices for Statistics*. Clarendon Press, Oxford, 2nd edition, 2000.

Nicholas J. Higham. *Functions of Matrices: Theory and Computation*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2008. ISBN 978-0-898716-46-7.

R. R. Joaquim, A. Martins, Peter Sturdza, and Juan J. Alonso. The Connection Between The Complex-Step Derivative Approximation And Algorithmic Differentiation. `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.12.8097`, 2001.

Steven G. Johnson. The NLopt nonlinear-optimization package, version 2.2.1. URL `http://ab-initio.mit.edu/nlopt`.

Tosio Kato. *Perturbation Theory for Linear Operators*. 1966.

Joze Korelc. Direct computation of critical points based on crout's elimination and diagonal subset test function. *Computers & Structures*, 88(3-4):189 – 197, 2010. ISSN 0045-7949. doi: DOI:10.1016/j.compstruc.2009.10.001. URL `http://www.sciencedirect.com/science/article/B6V28-4XNMBSS-3/2/a7084bc877c220706190e31ad6ea3267`.

S. Körkel, Arellano-Garcia H., J. Schöneberger, and G. Wozny. Optimum experimental design for key performance indicators. In *18th European Symposium on Computer Aided Process Engineering*, 2008.

Stefan Körkel. *Numerische Methoden für optimale Versuchsplanungsprobleme bei nichtlinearen DAE-Modellen*. PhD thesis, Universität Heidelberg, 2002.

Ekaterina Kostina, Michael A. Saunders, and Inga Schierle. Computation of Covariance Matrices for Constrained Parameter Estimation Problems Using LSQR, 2009. URL `http://www.uni-marburg.de/fb12/forschung/berichte/berichtemathe/pdfbfm/bfm09-01`.

Künsch. Mathematische Statistik (lecture script), 2005.

Waelbroeck Lucien. A polarization formula. In *Topological Vector Spaces and Algebras*, volume 230 of *Lecture Notes in Mathematics*, pages 118–133. Springer Berlin / Heidelberg, 1971. URL http://dx.doi.org/10.1007/BFb0061242. 10.1007/BFb0061242.

Jan R. Magnus and Heinz Neudecker. *Matrix differential calculus with applications in statistics and econometrics*. John Wiley & Sons, 2nd edition, 1999.

Joaquim R. R. A. Martins, Peter Sturdza, and Juan J. Alonso. The complex-step derivative approximation. *ACM Trans. Math. Softw.*, 29(3):245–262, 2003. ISSN 0098-3500. doi: http://doi.acm.org/10.1145/838250.838251.

Roger Mead. *The Design of Experiments: Statistical Principles for Practical Applications*. Cambridge University Press, July 1990. ISBN 0521287626. URL http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0521287626.

Marina Menshikova. *Uncertainty Estimation Using the Moments Method Facilitated by Automatic Differentiation in Matlab*. PhD thesis, Cranfield University, January 2010.

Uwe Naumann. Optimal accumulation of jacobian matrices by elimination methods on the dual computational graph. *Math. Program.*, 99(3):399–421, 2004.

Uwe Naumann. Dag reversal is np-complete. *J. Discrete Algorithms*, 7(4):402–410, 2009.

Richard D. Neidinger. Directions for computing truncated multivariate Taylor series. *Mathematics of Computation*, 74(249):321–340, 2005.

J.C. Newman, W.K. Anderson, and D.L. Whitfield. Multidisciplinary sensibility derivatives using complex variables. Technical report, Mississippi State University, 1998. URL http://fun3d.larc.nasa.gov/papers/MsReport.pdf. MSSU-COE-ERC-98-08.

Timothy Nguyen. A lower bound on the radius of analyticity of a power series in a real Banach space. *studia math.*, 191:171–179, 2009.

Travis E. Oliphant. *Guide to NumPy*. Trelgol Publishing, USA, 2006.

Eric Todd Phipps. *Taylor Series Integration of Differential-Algebraic Equations: Automatic Differentation as a Tool For Simulationg Rigid Body Mechanical Systems*. PhD thesis, Cornell University, February 2003.

Luc Pronzato. Survey paper: Optimal experimental design and some related control problems. *Automatica*, 44:303–325, February 2008. ISSN 0005-1098. doi: 10.1016/j.automatica.2007.05.016. URL http://portal.acm.org/citation.cfm?id=1330772.1330955.

F. Pukelsheim. *Optimal design of experiments*. Wiley series in probability and mathematical statistics. John Wiley & Sons, 1993.

Daniel Quinlan, Chunhua Liao, Thomas Panas, Robb Matzke, Markus Schordan, Rich Vuduc, and Qing Yi. Rose User Manual. http://rosecompiler.org, Version 0.9.5a, 2011.

Arno Rasch and H. Martin Bücker. Efcoss: An interactive environment facilitating optimal experimental design. *ACM Trans. Math. Softw.*, 37:13:1–13:37, April 2010. ISSN 0098-3500. doi: http://doi.acm.org/10.1145/1731022.1731023. URL http://doi.acm.org/10.1145/1731022.1731023.

A. S. Schäfer. *Effiziente reduzierte Newton-änliche Verfahren zur Behandlung hochdimensionaler strukturierter Optimierungsprobleme mit Anwendung bei biologischen und chemischen Prozesssen*. PhD thesis, Ruprecht-Karls-Universität Heidelberg, December 2004.

J. P. Schlöder. *Numerische Methoden zur Behandlung Hochdimensionaler Aufgaben der Parameteridentifizierung*. PhD thesis, Rheinische Friedrich-Wilhelms-Universität zu Bonn, 1987.

James R. Schott, editor. *Matrix Analysis for Statistics*. Wiley, New York, 1997.

George A. F. Seber. *A Matrix Handbook for Statisticians*. Wiley-Interscience, New York, NY, USA, 2007. ISBN 0471748692, 9780471748694.

Khodr Mahmoud Shamseddine. *New Elements of Analysis on the Levi-Civita Field*. PhD thesis, Michigan State University, 1999.

J. Shao. *Mathematical statistics. 2nd ed.* Springer Texts in Statistics. New York, NY, 2003.

S. P. Smith. Differentiation of the Cholesky Algorithm. *Journal of Computational and Graphical Statistics*, 4(2):134–147, 1995. URL `http://www.jstor.org/stable/1390762`.

G. W. Stewart. *Matrix Algorithms, Volume 1: Basic Decompositions*. SIAM, 1st edition, 1998.

Krister Svanberg. The method of moving asymptotes-a new method for structural optimization. *International Journal for Numerical Methods in Engineering*, 24:359 – 373, February 1987.

Krister Svanberg. A class of globally convergent optimization methods based on conservative convex separable approximations. *SIAM Journal on Optimization*, 12(2):555–573, 2002. doi: 10.1137/S1052623499362822. URL `http://link.aip.org/link/?SJE/12/555/1`.

SymPy Development Team. Sympy: Python library for symbolic mathematics, 2009. URL `http://www.sympy.org`.

Erik G.F. Thomas. A polarization identity for multilinear maps. `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.53.3481&rep=rep1&type=pdf`.

N.P. van der Aa, H.G. Ter Morsche, and R.R.M. Mattheij. Computation of eigenvalue and eigenvector derivatives for a general complex-valued eigensystem. *Electronic Journal of Linear Algebra ISSN 1081-3810*, 16:300–314, 2007.

William J. Vetter. Matrix calculus operations and taylor expansions. *SIAM Review*, 15(2): 352–369, 1973. doi: 10.1137/1015034. URL `http://link.aip.org/link/?SIR/15/352/1`.

Sebastian F. Walter. Source Code of the Performance Comparison Between Univariate Taylor Polynomial Arithmetic at Different Hierarchical Levels. URL `http://github.com/b45ch1/hpsc_hanoi_2009_walter`.

Sebastian F. Walter. ALGOPY: algorithmic differentiation in python, 2009. URL `http://github.com/b45ch1/algopy`.

Sebastian F. Walter. Taylorpoly, 2010. URL `http://github.com/b45ch1/taylorpoly`.

Timothy Hugh Waterhouse. *Optimal Experimental Design For Nonlinear and Generalized Linear Models*. PhD thesis, University of Queensland, 2005.

R. Clint Whaley, Antoine Petitet, and Jack J. Dongarra. Automated empirical optimization of software and the ATLAS project. *Parallel Computing*, 27(1–2):3–35, 2001. Also available as University of Tennessee LAPACK Working Note #147, UT-CS-00-448, 2000 (`www.netlib.org/lapack/lawns/lawn147.ps`).

Wikipedia. Paris gun. URL `http://en.wikipedia.org/wiki/Paris_gun`.

Emanuel Winterfors and Andrew Curtis. Numerical detection and reduction of non-uniqueness in nonlinear inverse problems. *Inverse Problems*, 24(2):025016, 2008. URL `http://stacks.iop.org/0266-5611/24/i=2/a=025016`.

C. Xin, Q. Houduo, Q. Liqun, and T. Kok-Lay. Smooth convex approximation to the maximum eigenvalue function. 2004.

# List of Figures

# List of Tables

# Selbständigkeitserklärung

Ich erkläre, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Berlin, den 06.04.2011 Sebastian F. Walter